Informed Equation Learning

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Matthias Werner, M. Sc.
aus Ochsenhausen

Tübingen
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	10. Oktober 2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Georg Martius
2. Berichterstatter:	Prof. Dr. Jakob Macke

Disclaimer: this thesis uses Felix Dangel's thesis-template that is based on Federico Marotta's `kaobook` template.

Acknowledgments

First of all, I would like to thank Prof. Dr. Philipp Hennig, Prof. Dr. Georg Martius and Dr. Andrej Junginger for making my doctorate possible. Thank you for your guidance, support and trust throughout my PhD.
Thank you Philipp for your always friendly, helpful and caring supervision and for allowing me to work independently and pursue my own ideas. I greatly admire your outstanding expertise and extensive experience in research, with which you have not only supported me, but also many other doctoral students, and introduced them to the field of research.
Thank you Georg for your motivating and insightful supervision and for your always positive support in completing this thesis. Your enthusiasm for the subject was throughout palpable.
Thank you Andrej for your guidance and endless efforts to connect my research with applications whenever I needed support. You were there to help and advise me. I really appreciate your always friendly, open and helpful manner.

I would like to thank Prof. Dr. Jakob Macke for his commitment as an examiner, who put a lot of time and effort into the evaluation of this thesis.

MPI-IS, the University of Tübingen and ETAS (Bosch GmbH) have offered me a great working environment in which I have been able to develop both academically and personally. I would like to thank all my colleagues and friends for the inspiring academic but also private dialogs and the open, humorous and collegial atmosphere, where I have been able to learn and work for the last few years. I would not have made it this far without all of you.
I would also like to thank my colleagues from the *Methods of Machine Learning* group at University of Tübingen for welcoming me in such an open and friendly way and for all the wonderful moments spent playing table football and enjoying coffee, especially: Philipp Hennig, Filip de Roos, Felix Dangel, Alexandra Gessner, Frank Schneider, Simon Bartels, Lukas Balles, Maren Mahsereci, Motonobu Kanagawa, Hans Kersting and Agustinus Kristiadi.
It was a great pleasure to work with my colleagues from the *Machine Learning Team* at ETAS, especially Andrej Junginger, Michael Öchsle, Damian Boborzi, Thilo Strauss, Markus Hanselmann, Sebastian Boblest, Jens Buchner, Holger Ulmer and Michael Ungermann.

I really enjoyed our weekly rides from Stuttgart to the University of Tübingen with Filip de Roos, Olga Lukashova-Sanz and Michael Öchsle, even though we always left way too early in the morning. In this sense, also many thanks to Christian Benz for your free couch, which saved me several times.

Thanks for proofreading parts of this manuscript Christian Rohwer, Felix Dangel, Filip de Roos, Georg Martius, Michael Lohaus, Dorea Wink and Philipp Arras.

A special thanks to my friends, who I got to know and to cherish during my physics studies, Kai Guther, Lukas Fischer, Jonas Heverhagen, Maximilian Mußotter, Matthias Pöhnl, Ruben Pöhnl and Toni Peter. Without you and a good dose of nerd talk, I would never be where I am now.

I would like to thank all my friends that helped me to keep a healthy balance during my PhD especially all the musicians who enabled those amazing projects with the chamber choir *Stuttgart Vokal*, the chamber orchestra *Junge Kammerphilharmonie Rhein-Neckar*, the chamber ensemble *Schwäbisches Salonensemble* and several Jazz-formations.

I would also like to take this opportunity to express my deepest gratitude to my family, in particular my mum Dorothea, my dad Hubert, my sister Julia, my brother David and my grandma Beate. You have always supported me and offered me a safe and loving support. You have challenged me, encouraged me to try new things and not shy away from challenges. I would like to thank you with all my heart for all your help and advice. In times of lack of motivation, it is not always easy to push ahead with research and complete a thesis. Thank you very much, for pushing me at the right moment and having my back. The regular childcare provided by my mother-in-law, Carola Wink, was an extraordinary help especially in the final phase of my thesis, thank you for that.

I would also like to thank my wife Dorea from the bottom of my heart. Without you, your daily support, your words of encouragement and your loving affection, I would not be the person I am today. Together with our children, Juna and Jaron, we have already experienced so many wonderful moments of happiness and I look forward with confidence to the coming decades that we will experience together.

Matthias Werner
Stuttgart 2024

Abstract

Equations are not only key to describing phenomena and their underlying principles in the natural sciences, but also play an important role in the engineering domain, e.g., in model-predictive control or as components describing complex systems. The task of learning equation-based models in an automated fashion is referred to as symbolic regression. Within the broader field of interpretability, this approach is becoming increasingly important for machine learning, which mostly generates black-box models. This thesis studies a differentiable relaxation to symbolic regression called equation learning and proposes new deep learning algorithms for scaling it to realistic settings in science and engineering. The thesis is structured in three conceptual parts.

The first part describes strategies to enhance expressivity and to stabilize training. Important atomic functions such as logarithm and division have restricted domains and singularities that lead to unstable training, making them difficult to discover. Our robust training method enables networks to deal with such atomic functions with singularities, which is an important step towards real-world applications. Normally, the computation of the Pareto front requires a search over network architectures with different numbers of hidden layers. This extensive search can be avoided by adding copy units. Together with a probabilistic L_0 regularization scheme, these methods form the basis for incorporating domain and expert knowledge into equation learning, which is considered in more detail in the second conceptual part.

The second part focuses on incorporating expert knowledge into equation learning. Domain-specific knowledge provided by experts can, in principle, guide and accelerate the search for better equations. This is of particular interest for complex datasets with a huge search space of possible equations. We call the resulting approach *informed equation learning neural network* (iEQL). Expert knowledge is incorporated by prohibiting certain combinations of functions within the neural network architecture. By utilizing a user-dependent weighting scheme that favors certain types of functions, the search for equations during training can be guided. Its application to several artificial and real-world experiments from the engineering domain is studied. The iEQL is shown to learn plausible and interpretable models with high predictive power.

The third part deals with uncertainty quantification for equation learning. Inspired by the automatic statistician, simple but effective forms of Bayesian deep learning are used to build structured and interpretable uncertainty over a set of plausible equations. Thus, two components of uncertainty have been identified: (i) *global uncertainty*, given by the differences in structure of each equation, and (ii) *local uncertainty*, given by the parametric uncertainty within one equation structure. Specifically, a mixture of Laplace approximations is used. Each mixture component captures a different equation structure, and the local Laplace approximation captures the corresponding parametric uncertainty. The approach is applied to toy examples and two real-world datasets.

These advances collectively contribute to the overarching objective of enhancing symbolic regression methods to align with the demands of contemporary applications in industry and research.

Zusammenfassung

Gleichungen sind in den Naturwissenschaften unverzichtbar um Phänomene und deren zugrunde liegende Prinzipien zu beschreiben. Auch im Ingenieurwesen, zum Beispiel bei der modellprädiktiven Kontrolle, sind sie essenziell zur Darstellung komplexer Systeme. Das Erlernen solcher Modelle auf automatisierte Weise wird als Gleichungslernen bezeichnet. Im Forschungsbereich des Machinellen Lernens werden meist undurchsichtige Modelle geliefert. Hier gewinnt Gleichungslernen für die Erklärbarkeit dieser Modelle immer mehr an Bedeutung. Diese Arbeit stellt neue Deep-Learning-Algorithmen vor, um Gleichungslernen mit Neuronalen Netzen für reale Anwendungen in Wissenschaft und Technik zu skalieren. Diese Dissertation gliedert sich in drei konzeptionelle Teile.

Der erste umfasst Strategien zur Verbesserung der Ausdrucksfähigkeit und zur Stabilisierung des Trainings. Funktionen mit eingeschränkten Bereichen und Singularitäten, wie Logarithmen und Division, stellen beim Gleichungslernen eine Herausforderung dar, da diese zu instabilem Training führen können. Unsere robuste Trainingsmethode ermöglicht es den Netzen mit solchen atomaren Funktionen, die Singularitäten besitzen, umzugehen. Die Methode ist daher ein wichtiger Schritt in Richtung realer Anwendungen. Normalerweise erfordert die Berechnung der Pareto-Front eine Suche über Netzwerkarchitekturen mit unterschiedlicher Anzahl von versteckten Schichten. Durch Hinzufügen von Kopiereinheiten kann diese umfangreiche Suche vermieden werden. Zusammen mit einer probabilistischen L_0 Regularisierung bilden diese Methoden die Basis zur Einbindung von Fach- und Expertenwissen in das Gleichungslernen.

Der zweite Teil konzentriert sich auf die Einbindung von Fach- und Expertenwissen in das Gleichungslernen. Expertenwissen kann die Suche nach besseren Gleichungen leiten und beschleunigen, was für komplexe Datensätze essenziell ist. Das von uns entwickelte gleichungslernende neuronale Netz (iEQL) schließt bestimmte Kombinationen von Funktionen aus und verwendet ein benutzerabhängiges Gewichtungsschema um bestimmte Funktionstypen während des Trainings zu bevorzugen. Die Anwendung des Verfahrens wird anhand mehrerer künstlicher und realer Datensätze aus dem Ingenieurwesen demonstriert, in denen der iEQL plausible und erklärbare Modelle mit hoher Vorhersagekraft gelernt hat.

Der dritte Teil befasst sich mit Unsicherheitsquantifizierung für das Gleichungslernen. Inspiriert durch den automatischen Statistiker, werden einfache, aber effektive Formen des Bayes'schen Deep Learnings verwendet, um eine strukturierte und erklärbare Unsicherheit über eine Reihe plausibler Gleichungen aufzubauen. Daher werden zwei Komponenten der Unsicherheit identifiziert (i) globale Unsicherheit, bedingt durch strukturelle Unterschiede der Gleichungen und (ii) lokale Unsicherheit, bedingt durch parametrische Unsicherheit innerhalb einer Gleichungsstruktur. Die identifizierte Unsicherheit wird mit einer Kombination aus Laplace-Approximationen modelliert, wobei jede Komponente eine andere Gleichungsstruktur erfasst, und die lokale Laplace-Approximation erfasst jeweils die parametrische Unsicherheit einer Gleichung. Seine Anwendung wird anhand von künstlichen Beispielen und zwei realen Datensätzen demonstriert.

Diese Fortschritte tragen gemeinsam zu dem übergeordneten Ziel bei, symbolische Regression so zu verbessern, dass sie den Anforderungen moderner Anwendungen in Industrie und Forschung gerecht wird.

Table of Contents

Notation

$\boldsymbol{\theta}$	Parameter vector
$\boldsymbol{W}^l$	Weight matrix of (hidden) layer l
$\boldsymbol{b}^l$	Bias vector of (hidden) layer l
$\boldsymbol{z}^l$	Output of (hidden) layer l before a non-linearity is applied
$\boldsymbol{h}^l$	Output of (hidden) layer l
L	Total number of layers
$f_{\boldsymbol{\theta}}$	Function parameterized by set of parameters $\boldsymbol{\theta}$
$\mathcal{L}$	Loss term
$\mathcal{D}$	Dataset
$\mathcal{D}^{\text{train}}$	Train dataset
$\mathcal{D}^{\text{valid}}$	Validation dataset
$\mathcal{D}^{\text{valid}}_{\text{int}}$	Validation dataset for interpolation
$\mathcal{D}^{\text{valid}}_{\text{ex}}$	Validation dataset for extrapolation
$\mathcal{D}^{\text{test}}$	Test dataset
$\mathcal{D}^{\text{test}}_{\text{int}}$	Test dataset for interpolation
$\mathcal{D}^{\text{test}}_{\text{ex}}$	Test dataset for extrapolation
$\text{V}_{\text{int}}\text{-S}$	Selection criterion based on complexity and validation accuracy
$\text{V}_{\text{int\&ex}}$	Selection criterion based on extrapolation and validation accuracy
V_{int}	Selection criterion solely based on validation accuracy
$\text{Ber}(\pi)$	Bernoulli distribution with dropout rate $1-\pi$
$\mathcal{U}(a,b)$	Uniform distribution over the interval (a,b)
$\mathcal{N}(x \mid \mu, \sigma^2)$	Normal/Gaussian distribution of random variable x, with mean μ, variance σ^2, and density $1/\sigma\sqrt{2\pi}\exp((x-\mu)^2/2\sigma^2)$
$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multi-variate normal/Gaussian distribution of random vector $\boldsymbol{x}$ with mean vector $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$, and density $1/(\sqrt{2\pi\det\boldsymbol{\Sigma}})\exp(-1/2(\boldsymbol{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}))$
BFGS	Broyden-Fletcher-Goldfarb-Shanno algorithm
CNN	Convolutional neural network
EQL	Equation learning neural network [62]
EQL$^{\div}$	Equation learning neural network with final division layer [72]
GGN	Generalized Gauss-Newton (matrix)
GP	Gaussian process
iEQL	Informed equation learning neural network [84]
L-BFGS	Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm
MAP	Maximum a posteriori (estimation)
MLP	Multi-layer perceptron
MP	Mean predictor on the train datatset
MSE	Mean squared error
PySR	Genetic algorithm by [14]
ReLU	Rectified linear unit
RMSE	Root mean squared error
MoLA	Mixture of Laplace approximations

Overview 1

1.1 Introduction

The natural sciences rely heavily on equations to describe phenomena and their underlying principles. The search for equations that describe and explain the underlying principles of new phenomena that can not be explained with the current state of knowledge, is part of the daily life of scientists. Physics, in particular, uses equations to build a coherent description of natural laws. For example, Newton's and Kepler's laws or Schrödinger's equation describe and encode the phenomena of motion, celestial dynamics [23], or quantum mechanics [2], respectively. Those equations are typically derived from fundamental principles such as the conservation laws or phenomenological approaches. Similarly, they are important in the engineering domain, e.g., in model-predictive control [25]. The designed equations then correspond to a hypothesis for the inner system's behavior and represent its relations and properties. In industrial applications, e.g., for embedded controller, models have to be minimal in computational power and memory demand due to embedded hardware and latency constraints. Model predictions given by equations can meet those requirements.

[23] Feynman, Leighton, and Sands (1965), "The feynman lectures on physics"

[2] Ballentine (1970), "The Statistical Interpretation of Quantum Mechanics"

[25] García, Prett, and Morari (1989), "Model predictive control: Theory and practice—A survey"

Due to a continuous increase of measurement data in science and various engineering disciplines there is a growing demand for automated equation discovery to model and understand the underlying systems. Recent progress in machine learning, computational resources and data availability pushes the boundaries of data-driven discovery of equations. Learning those equations in an automated fashion from data is referred to as *symbolic regression*. In contrast to traditional regression or supervised deep learning, which only optimizes a set of parameters for prediction, symbolic regression simultaneously discovers the structure of the equation and its numerical coefficients. These equations can be defined by combining and linking *mathematical operators*[1], *mathematical functions*[2] and *numerical constants*. Common requirements for symbolic regression are

1: mathematical operators: e.g. multiplication, division, partial derivatives, integrals

2: mathematical functions: e.g. exponential, sine, cosine, logarithm

- *Interpretability*: The structure and complexity associated with the equation should be as simple as possible, given the ability of an expert in the field to interpret the equation.
- *Generalization*: The equation must capture the underlying properties of the system, such that it can generalize and extrapolate to unseen data.

Algorithms for symbolic regression aim to optimize for both interpretability and generalization. This enables handling numerically similar expressions on a bounded set of datapoints[3], as well as incorporating mathematical identities[4]. To identify the underlying mathematical expression of a system, it is not sufficient to optimize solely for accuracy of the predictions. Therefore, symbolic regression commonly obeys the principle of Occam's razor[5] and must trade-off model complexity against accuracy. This approach leads to a multi-objective optimization challenge for accuracy and simplicity. Symbolic regression algorithms must calculate Pareto optimal expressions that cannot improve an objective without sacrificing performance on another objective.

3: similar expressions e.g. Taylor-Series

$$\sinh x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \mathcal{O}\left(\frac{x^7}{7!}\right)$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \mathcal{O}\left(\frac{x^5}{5!}\right)$$

4: mathematical identities e.g.:

$$\sin^2(x) + \cos^2(x) = 1$$

$$\sinh x = (e^x + e^{-x})/2$$

5: William of Occam 1285-1349: "plurality should not be assumed without necessity"

The conventional approach to symbolic regression tasks typically involves discrete search methods such as genetic programming [13] and evolutionary algorithms [41, 48]. These methods are powerful for small-scale problems and mimic natural biological evolution by generating a series of computer programs, evaluating their effectiveness, and evolving the most promising programs. A notable example of success is the automated discovery of natural laws by Schmidt and Lipson [73]. But with advances in machine learning and in particular deep learning, it has become possible to tackle more complex challenges in robotics and industry where these traditional methods reach their limits. For instance, Biggio et al. [3] utilized set-transformers trained on numerous equation-dataset pairs to refine the search for optimal equations.

[13] Cramer (1985), "A Representation for the Adaptive Generation of Simple Sequential Programs"

[41] Koza (1992), "On the programming of computers by means of natural selection, Genetic Programming, vol. 1"

[48] Langdon, Poli, McPhee, and Koza (2008), "Genetic programming: An introduction and tutorial, with a survey of techniques and applications"

[73] Schmidt and Lipson (2009), "Distilling Free-Form Natural Laws from Experimental Data"

[3] Biggio, Bendinelli, Neitz, Lucchi, and Parascandolo (2021), "Neural Symbolic Regression that scales"

Further innovations include the use of gradient information during training. This concept was explored in a reinforcement learning context by Petersen et al. [67] using risk-seeking policy gradients. A particularly effective approach utilizing gradient information are *equation learning neural networks* (EQL) introduced by Martius and Lampert [62] . They encode complex equations within their architecture, employing various activation functions like sine, cosine, multiplication and division, across hidden layers. Throughout training, the EQL gradually omits irrelevant components, converging on a sparse representation that effectively captures the desired equation. This design not only enables seamless integration into larger deep learning architectures for comprehensive end-to-end training but also adapts well to large datasets, leveraging recent advancements in machine learning. This scalability and integration capability enhance the utility of EQL in tackling complex equations and broadening the scope of applications across various fields such as the discovery of partial differential equations [53] and integration with convolutional networks for digit extraction in arithmetic tasks [38]. Additionally, Lin et al. [51] demonstrate the effectiveness of EQL in modeling energy functions based on dynamic observations within the field of density functional theory.

[67] Petersen, Larma, Mundhenk, Santiago, Kim, and Kim (2021), "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients"

[62] Martius and Lampert (2016), "Extrapolation and learning equations"

Exploring the equation learning framework is particularly compelling for tackling real-world problems characterized by high-dimensional data and complex, nonlinear relationships. Unlike traditional methods that might struggle with the scale and diversity of modern datasets. This thesis builds on equation learning neural networks introduced by Martius and Lampert [62].

[62] Martius and Lampert (2016), "Extrapolation and learning equations"

The present work proposes new deep learning algorithms to scale equation learning neural networks to realistic settings in science and engineering. To tackle this challenge, this thesis addresses three related research questions:

Q1 How to broaden the expressivity of equation learning neural networks and train them efficiently?
Q2 How to utilize domain knowledge to guide the search for better equations?
Q3 How to quantify uncertainty of a set of equations that was discovered by an equation learning neural network?

research question 1

In general, a large expressivity of the equation learner is desired to learn complex equations. Yet, common functions like logarithm, square root or division are hard to discover with equation learning neural networks. Their restricted domains and singularities lead to highly unstable training. This motivates the research question *"How to broaden the expressivity of equation learning neural networks and train them efficiently?"* Our scientific contribution to address this question is presented in chapter 4. It includes a stable training procedure based on a learnable relaxation of the singularities as well as a specific L_0 regularization scheme.

research question 2

As a result of the great expressive power of the equation learner, an exhaustive search in the hypothesis space is required. Prior knowledge in form of domain specific knowledge provided by experts is an important source of information to scale equation learning to realistic settings in science and engineering. Yet, its application to equation neural network is challenging and has not been extensively explored. This leads to the research question *"How to utilize domain knowledge to guide the search for better equations?"*. In order to address this question we introduce an informed equation learning neural network (iEQL) in chapter 5. It incorporates expert knowledge about what are permitted or prohibited equation components, as well as a domain dependent structured sparsity prior to improve the search for accurate and simple equations. We demonstrate applications of this equation learner to several artificial and real-world experiments.

research question 3

Equation learners discover a set of plausible equations with different structure and varying complexity. Their differences in structure can lead to a rich variety of predictions. Choosing one equation out of the others by chance might lead to overconfident predictions. In order to apply equations to safety critical systems like health care and automated driving it is crucial to know about their uncertainty. This challenge is tackled by exploring the research question *"How to quantify uncertainty of a set of equations that was discovered by an equation learning neural network?"*. As a means of addressing this question two components of uncertainty are identified in chapter 6 (i) *global uncertainty* given by the differences in structure of each equation and (ii) *local uncertainty* given by the parametric uncertainty within one family of equations. We propose a mixture of Laplace approximations to capture both types of uncertainty within one model and demonstrate its applications to artificial and real-world experiments.

1.2 Outline

This thesis is structured into three main parts: Part I describes relevant principles and concepts necessary to understanding this thesis. Part II explores and elaborates on the scientific contributions. Part III examines the implications of these findings and suggests directions for future research.

The first part I deals with important machine learning components in chapter 2 and symbolic regression in chapter 3.

There is a brief introduction to Bayesian inference in section 2.1. Section 2.2 discusses regularized empirical risk and in section 2.3 both concepts are compared. At the end of the first chapter section 2.4 briefly introduces deep neural networks.

Chapter 3 refers to symbolic regression in general and specifically to equation learning neural networks. Section 3.1 gives a detailed introduction to symbolic regression and different approaches to this problem. Section 3.2 introduces equation learning as a differentiable relaxation of symbolic regression that allows applying continuous search methods in contrast to the classical discrete search algorithms that are usually used for symbolic regression. Section 3.3 presents the general architecture of equation learning neural networks, which form the basis of this thesis. Special features and important properties of the training of such neural networks are discussed in section 3.4. And the last section 3.5 discusses different model selection criteria used in Equation Learning.

Part II discusses and presents our contributions to the three research questions Q1, Q2 and Q3 in chapter 4, chapter 5 and chapter 6 respectively.

Chapter 4 deals with the extension of the expressive power of equation learning neural networks and training stability. Section 4.1

provides an introduction to the topic and section 4.2 gives a comprehensive overview of the related literature. Section 4.3 presents an extension of the expressive power of equation learning neural networks to atomic functions with singularities, such as division or logarithm. An extension of the Equation Learner architecture with copying units is proposed in section 4.4. This approach eliminates the need for an exhaustive search for the optimal number of hidden layers. Section 4.5 discusses a probabilistic, differentiable L_0 regularization scheme that does not cause parameter shrinkage during the training process. Section 4.6 summarizes the presented methods and establishes the link to chapter 5.

Chapter 5 deals with the inclusion of expert- and domain knowledge in equation learning. Strategies for integrating expert knowledge into both the training process and the architecture of the equation learner are investigated. Section 4.1 motivates the considered challenges. Section 5.2 presents a method that can incorporate expert knowledge about allowed or disallowed equation components into the neural network architecture. Section 5.3 explores domain-specific structured sparsity priors, aimed at refining the search for accurate and straightforward equations. Both methods are combined in section 5.4 in the informed Equation Learner (iEQL) to adapt the application to realistic conditions in science and engineering. Section 5.5 presents several artificial and real-world engineering experiments in which our system develops interpretable models with high prediction accuracy. The chapter ends with a summary of the main results and contributions in section 5.6.

Chapter 6 is dedicated to quantifying the uncertainty in equation learning. Inspired by the approach of the automatic statistician, we use simple yet effective methods of Bayesian deep learning to develop structured and interpretable uncertainty across various plausible equations. Section 6.1 focuses on two main aspects of uncertainty: (i) global uncertainty, which arises from structural differences between individual equations, and (ii) local uncertainty, which stems from parametric uncertainty within a family of equations. Section 6.2 provides an overview of related research in the area of uncertainty quantification and equation learning. Section 6.3 presents two approaches from a Bayesian perspective to capture both global and local uncertainty in equation learning. Section 6.4 demonstrates the application of our method to two artificially generated ambiguous datasets and two real-world datasets. The main results and key contributions are summarized in section 6.5.

Part III provides a summary, conclusion, and an outlook on future research directions. Section 7.1 delivers a detailed summary of the thesis's main contributions, linking them to the initial research questions Q1, Q2, and Q3 introduced in section 1.2. The section concludes by outlining future research directions in section 7.2.

Disclaimer 1.1 Chapter 4 and chapter 5 are based on a preprint with the following co-author contributions:

M. Werner, A. Junginger, P. Hennig, and G. Martius. "Informed Equation Learning". *arXiv: 2105.06331* (2021) [84]

	Ideas	**Experiments**	**Analysis**	**Writing**
M. Werner	60 %	75 %	60 %	60 %
A. Junginger	10 %	10 %	10 %	10 %
P. Hennig	5 %	5 %	10 %	10 %
G. Martius	25 %	10 %	20 %	20 %

Disclaimer 1.2 Chapter 6 is based on the peer-reviewed workshop paper with the following co-author contributions:

M. Werner, A. Junginger, P. Hennig, and G. Martius. "Uncertainty in equation learning". *Genetic and Evolutionary Computation Conference Companion, GECCO, Workshop Proceedings*. 2022 [85]

	Ideas	**Experiments**	**Analysis**	**Writing**
M. Werner	65 %	80 %	75 %	70 %
A. Junginger	5 %	5 %	5 %	5 %
P. Hennig	20 %	5 %	10 %	10 %
G. Martius	10 %	10 %	10 %	15 %

List of Publications

Below you will find a list of all publications and patents that I co-authored during my PhD, whether they are relevant to this thesis or not.

M. Werner, P. Margaretti, and A. Maciołek. "Drag Force for Asymmetrically Grafted Colloids in Polymer Solutions". *Frontiers in Physics* (2019) [87]

F. Groß, M. Zelent, A. Gangwar, S. Mamica, P. Gruszecki, M. Werner, G. Schütz, M. Weigand, E. J. Goering, C. H. Back, M. Krawczyk, and J. Gräfe. "Phase resolved observation of spin wave modes in antidot lattices". *Applied Physics Letters* (2021) [28]

M. Werner, A. Junginger, P. Hennig, and G. Martius. "Informed Equation Learning". *arXiv: 2105.06331* (2021) [84]

F. Groß, M. Weigand, A. Gangwar, M. Werner, G. Schütz, E. J. Goering, C. H. Back, and J. Gräfe. "Imaging magnonic frequency multiplication in nanostructured antidot lattices". *Phys. Rev. B* (1 2022) [27]

M. Werner, A. Junginger, P. Hennig, and G. Martius. "Uncertainty in equation learning". *Genetic and Evolutionary Computation Conference Companion, GECCO, Workshop Proceedings*. 2022 [85]

Patents:

T. Strauss, M. Werner, A. Junginger, M. Hanselmann, H. Ulmer, and K. Dormann. "Method and device for training and producing an artificial neural network". Patent WO2020193481A1. 2020 [76]

M. Werner, A. Junginger, P. Hennig, G. Martius, and M. Hein. "Apparatus and method for estimating uncertainties". Patent DE102021124928A1. 2023 [86]

Part I

Background & Motivation

Machine Learning Components 2

This chapter provides the theoretical basics of necessary machine learning components for this thesis. A short introduction to Bayesian inference is given in section 2.1. Section 2.2 introduces regularized empirical risk and its Bayesian interpretation is discussed in section 2.3. An overview of deep neural networks is given in section 2.4

2.1 Bayesian Inference

This section gives a brief and thus also incomplete introduction to probabilistic machine learning. It focuses on the necessary concepts relevant to this work. For a complete treatment of probability theory and Bayesian statistics please refer to Jaynes [36] and MacKay [57].

[36] Jaynes (2003), "Probability theory: The logic of science"

[57] MacKay, Mac Kay, et al. (2003), "Information theory, inference and learning algorithms"

Following the argument of Cox [12], degrees of belief about states in the world can be mapped onto probabilities if they satisfy certain consistency rules. Thus, in the context of Bayesian statistics, probabilities are used to determine degrees of belief of such states as well as of inferences that are based on such states.

[12] Cox (1946), "Probability, frequency and reasonable expectation"

In the context of machine learning, typically, a probabilistic model describes the relation between observations, here called *data* ($\mathcal{D}$) , and unknown quantities such as model parameters, model structure, latent variables or predictions at unobserved locations, here called *hypothesis* in short h. In general, with the help of *Bayes' rule* (e.g. MacKay [57]) the conditional probability given the observations can be calculated as

Bayes' rule

$$p(\text{hypothesis} \mid \text{data}) = \frac{p(\text{data} \mid \text{hypothesis}) \times p(\text{hypothesis})}{\sum_h p(\text{data} \mid h) p(h)} \tag{2.1}$$

The initial distribution of the *hypothesis* is called *prior* and describes the initial state of believe about the hypotheses before the observations. The *likelihood* describes how likely the observations are given a particular hypothesis. The conditional probability of equation (2.1) on the left-hand side is called *posterior* and is not known in advance. It describes the state of believe about the hypotheses after the observations. It can be calculated from known quantities with the right-hand side of equation (2.1). The *evidence*, also called marginal-likelihood, normalizes the posterior. It does not depend on a specific hypotheses and is thus not relevant to determine the

prior

likelihood

posterior

evidence

relative probabilities of different hypotheses. Knowing all those terms, equation (2.1) can be generally described as

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} .$$

In the domain of machine learning, hypothesis abstraction is typically tiered in to two levels, with the model (m) being instantiated through its parameters (θ). Within this framework, Bayesian inference serves as a fundamental mechanism for updating our belief about the parameters given new data. The posterior distribution $p(\theta \mid \mathcal{D}, m)$ is computed by applying Bayes' theorem, which relates the posterior to the likelihood $p(\mathcal{D} \mid \theta, m)$ and the prior $p(\theta \mid m)$ over the parameters, normalized by the evidence $p(\mathcal{D}|m)$ as follows

$$p(\theta \mid \mathcal{D}, m) = \frac{p(\mathcal{D} \mid \theta, m) \times p(\theta \mid m)}{p(\mathcal{D} \mid m)} . \tag{2.2}$$

This formulation asserts that the updated belief about the parameters (posterior) is proportional to our prior belief adjusted by the information provided by the observed data (likelihood).

predictive distribution

In many practical scenarios, the interest lies in the *predictive distribution* that is evaluating the posterior distribution of a model m at a new set of locations $\mathcal{D}^*$. It is an average of all possible parameter values, which are weighted by the probability of their posterior

$$p(\mathcal{D}^* \mid \mathcal{D}, m) = \int p(\mathcal{D}^* \mid \theta, m) p(\theta \mid \mathcal{D}, m) \, \mathrm{d}\theta . \tag{2.3}$$

Furthermore, the integral of the likelihood of the data given the parameters, weighted by the prior over the parameters, yields the model evidence $p(\mathcal{D}|m)$. This model evidence is pivotal for Bayesian model comparison, as it inherently integrates out the parameter uncertainty

$$p(\mathcal{D} \mid m) = \int p(\mathcal{D} \mid \theta, m) p(\theta \mid m) \, \mathrm{d}\theta . \tag{2.4}$$

Bayesian model selection

When Bayesian inference is employed for model selection rather than parameter estimation, Bayes' rule is applied to models as a whole. The posterior probability of a model given the data is proportional to the product of the likelihood of the data under the model and the prior probability of the model, normalized by the evidence of the data

$$p(m \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid m) \times p(m)}{p(\mathcal{D})}$$

The evidence of the data $p(\mathcal{D})$ is obtained by summing over the model space, which involves calculating the model evidence for each model weighted by its prior probability

$$p(\mathcal{D}) = \sum_m p(\mathcal{D} \mid m) p(m)$$

In the case of Bayesian model selection it is sufficient to compare relative posterior probabilities and thus the evidence $p(\mathcal{D})$ is not necessary.

2.2 Regularized Empirical Risk

In the domain of machine learning, the performance of a model is often defined in terms of a loss or risk function and a regularization of the parameters. The regularization typically arises due to additional constraints or penalties on the model parameters to prevent overfitting or improve generalization of the model to new, unseen data. The expected loss, denoted as $\mathcal{L}$, quantifies the cost associated with the predictions made by the model and the true data distribution. Machine learning models strive to minimize the *regularized expected loss* as part of their learning process. It can be expressed as an expectation over the data distribution $\mathcal{Q}$, with the risk function l and a regularization g

expected loss

$$\mathcal{L} = g(\theta) + \mathbb{E}_{d \sim \mathcal{Q}} \left[l(d, m_\theta) \right] . \tag{2.5}$$

However, in practice, the true data distribution is rarely available due to computational constraints or the impossibility of accessing the entire data distribution. Thus, computing the expectation directly is not feasible. As a practical workaround, the expected loss is approximated by *regularized empirical risk*, which is the average loss over a dataset $\mathcal{D}$ of size $|\mathcal{D}|$ with regularization g

regularized empirical risk

$$\mathcal{L} = g(\theta) + \frac{1}{|\mathcal{D}|} \sum_{d_i \in \mathcal{D}} l(d_i, m_\theta) . \tag{2.6}$$

The dataset $\mathcal{D}$ consists of samples assumed to be independent and identically distributed (iid.) from the data distribution $\mathcal{Q}$. The empirical risk is a surrogate for the expected loss, and its minimization corresponds to training the model to fit the data.

2.3 Bayesian Interpretation of Regularized Empirical Risk

The Bayesian framework offers an insightful interpretation of regularized empirical risk through *maximum a posteriori* (MAP)

maximum a posteriori

estimation. Under the assumption of conditionally independent data samples given the model parameters θ. The MAP estimation aligns closely with the concept of regularized empirical risk. Taking the negative logarithm of the posterior distribution a clear correspondence between prior and regularization as well as likelihood and loss function becomes apparent

$$\mathcal{L} = -\ln p(\theta) - \ln p(\mathcal{D} \mid \theta, m) + \text{const.}\,. \tag{2.7}$$

The term const. represents a constant that ensures normalization but does not affect the parameter estimation process. In this expression, the negative log prior, $-\ln p(\theta)$, serves as the regularization term[1]. It incorporates prior knowledge into the model and typically penalizes complexity, ensuring that the estimates of model parameters θ do not stray too far from the initial beliefs. On the other hand, the negative log likelihood, $-\ln p(\mathcal{D} \mid \theta, m)$, relates to the loss function, quantifying how well the model explains the observed data.

1: A Gaussian (Laplace) prior on the parameters corresponds to a L_2 (L_1) regularization.

However, not all loss functions and regularizations can be framed within this Bayesian interpretation. The negative logarithm of a probability distribution imposes constraints; for instance, the integral of the distribution must converge to ensure that it normalizes to a probability measure. This requirement may not hold for all forms of loss functions and regularizations. For example the *hinge loss* used in support vector machines can not be represented with a negative log likelihood [68].

[68] Rasmussen, Williams, et al. (2006), "Gaussian processes for machine learning"

2.4 Deep Neural Networks

Deep Neural Networks, also known as feedforward neural networks, constitute a fundamental building block in the field of deep learning. These networks serve as universal function approximators $f_\theta : x \longrightarrow y$ to map a given input $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ to a predicted output $\boldsymbol{y} \in \mathcal{Y} \subseteq \mathbb{R}^{d'}$. They are parameterized by a set of parameters $\theta \in \mathbb{R}^W$.

A typical architecture of deep neural networks is a composition of multiple layers, where each layer consists of a linear transformation defined by a weight matrix $\boldsymbol{W}$ and a bias vector $\boldsymbol{b}$. This transformation is followed by an element-wise non-linear function σ. The combination of a linear mapping and the subsequent non-linear activation is called *hidden layer* and is expressed mathematically as follows

hidden layer

$$\boldsymbol{h}^l = \sigma^l(\boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l)\,. \tag{2.8}$$

In this context, $\boldsymbol{h}^{l-1}$ represents the output of the previous layer or the input for $l = 1$ where l indexes the layers within the network or the output for $l = L$, where L is the total number of layers.

The output of these hidden layers can be interpreted as feature vectors or hidden representations. The non-linear function σ is typically referred to as activation function[2].

2: typical activation functions e.g.: sigmoid, ReLU, tanh

The design and architecture of neural networks is an area of active research and is often tailored to specific tasks. The architecture varies from traditional multi-layer perceptrons [71] (MLP) to more sophisticated designs such as transformers [82]. Other notable architectures include convolutional neural networks [49] (CNNs), which are predominantly utilized in image classification tasks, and recurrent neural networks [30, 32] (RNNs), which are frequently employed for their temporal data processing capabilities. A comprehensive exploration of the latest advancements in deep learning is available in the book "Deep Learning" [26].

[71] Rosenblatt (1963), "Principles of neurodynamics. Percetrons and the theory of brain mchanisms"

[82] Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin (2017), "Attention is All you Need"

[49] LeCun, Haffner, Bottou, and Bengio (1999), "Object Recognition with Gradient-Based Learning"

[30] Hochreiter and Schmidhuber (1997), "Long Short-Term Memory"
[32] Hopfield (1982), "Neural networks and physical systems with emergent collective computational abilities."

[26] Goodfellow, Bengio, and Courville (2016), "Deep Learning"

Symbolic Regression 3

This chapter introduces symbolic regression and respectively equation learning in the field of machine learning. Section 3.1 presents the basics of symbolic regression and section 3.2 summarizes a differentiable relaxation to symbolic regression called equation learning that is relevant for this thesis. Section 3.3 then presents a special neural network architecture that was developed for this task. Section 3.4 addresses specific training challenges. The chapter concludes with section 3.5, which outlines the criteria for model selection.

3.1 Introduction

Learning the governing equations from empirical data in an automated fashion, is referred to as *symbolic regression*. As the complexity of the system under consideration increases, the number of atomic units required to describe it also increases. This leads to an exponentially growing number of possible equation structures. Symbolic regression therefore requires efficient search algorithms to find "good" equations. The definition of a "good" equation is not well-defined and typically results in a Pareto optimal solution that trades off accuracy with complexity. As a thread within the wider area of interpretable AI, it is of increasing importance to machine learning, which mostly produces black box models. Symbolic regression can be divided into three different approaches [9]:

[9] Camps-Valls, Gerhardus, Ninad, Varando, Martius, Balaguer-Ballester, Vinuesa, Diaz, Zanna, and Runge (2023), "Discovering causal relations and equations from data"

- *Discrete search methods* such as genetic algorithms,
- *Continuous search methods* that deal with a differentiable relaxation of the discrete problem and
- *Inverse mapping of data to plausible equations*.

The following discussion is meant to provide a coherent overview of symbolic regression that has been recently summarized in an extensive review provided by Camps-Valls et al. [9].

Discrete Search Methods for Symbolic Regression

Discrete search methods for symbolic regression are commonly addressed with genetic programming [13], like evolutionary algorithms [41, 48] or other discrete search algorithms. The concept is based on the use of a search algorithm to identify computer programs that can solve specific problems. To do this, numerous

[13] Cramer (1985), "A Representation for the Adaptive Generation of Simple Sequential Programs"

[41] Koza (1992), "On the programming of computers by means of natural selection, Genetic Programming, vol. 1"

[48] Langdon, Poli, McPhee, and Koza (2008), "Genetic programming: An introduction and tutorial, with a survey of techniques and applications"

random programs are generated iteratively, their performance evaluated and the most effective ones selected. These selected programs are then recombined and randomly modified to form a new pool of candidates. This approach is based on the biological evolutionary process observed in nature, which shapes the genetic composition of living organisms. In symbolic regression, the equations or functions can be formed from combinations and links of input variables, mathematical operators, elementary functions and numerical constants. A major success was the automated discovery of natural laws by Schmidt and Lipson [73]. Their method is available as online service [16] formerly implemented in a tool called Eureqa [19].

[73] Schmidt and Lipson (2009), "Distilling Free-Form Natural Laws from Experimental Data"

[16] DataRobot Inc (2024), "Eureqa as part of DataRobot's service"

[19] Dubčáková (2011), "Eureqa: software review"

Another promising approach that is based on discrete search, called Feynman AI [80, 81], utilizes domain knowledge in form of proven physical methods, including unit calculations, symmetries and separability within the dataset. It outperformed Eureqa in a benchmark involving 100 equations from the Feynman lectures by successfully recovering all 100 equations, while Eureqa was only able to recover 71 equations.

[80] Udrescu, Tan, Feng, Neto, Wu, and Tegmark (2020), "AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity"

[81] Udrescu and Tegmark (2020), "AI Feynman: A physics-inspired method for symbolic regression"

Using a unique approach, *Deep Symbolic Regression (DSR)* [67] employs deep reinforcement learning to address symbolic regression as an exploration challenge. The method uses a recurrent neural network to generate expressions represented as depth graph traversal sequences, where the numerical constants of the expressions are optimized using the BFGS algorithm. DSR proposes a risk-seeking policy gradient to target best-case expressions instead of good expressions on average.

[67] Petersen, Larma, Mundhenk, Santiago, Kim, and Kim (2021), "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients"

Continuous Space Search as Differentiable Relaxation of Symbolic Regression

In order to use efficient continuous optimization algorithms, a dense set of possible functions is searched. This can be achieved by using a very complex, parameterized function that can represent all structures of the functions in question. The goal is to find a sparse representation of this complex function that best fulfills the trade-off between complexity and accuracy. Thus, both the equation structure and the values of the parameters can be optimized using sparsity-inducing methods in continuous space.

If the function of interest can be represented as a sparse linear combination of known building blocks, a method known as SINDy [7] (Sparse Identification of Dynamical Systems) can be applied. It is an efficient method for analyzing dynamical systems using sparse linear regression. This approach identifies differential equations from observed data. A similar concept was previously introduced

[7] Brunton, Proctor, and Kutz (2016), "Discovering governing equations from data by sparse identification of nonlinear dynamical systems"

in the FFX method by McConaghy [63] for the general symbolic regression problem. In SINDy, the input data is processed through a given library of basis functions and interaction terms to generate potential features. The linear combination of those features represents a complex function, which is then refined to a sparse representation using sparse linear regression techniques on the observed data, such as L_1 regularization (Lasso [78]).

[63] McConaghy (2011), "FFX: Fast, Scalable, Deterministic Symbolic Regression Technology"

[78] Tibshirani (1996), "Regression shrinkage and selection via the lasso"

The extension of the idea of sparse linear regression to nested building blocks can be achieved with the help of neural networks. A notable early example is the Equation Learner (EQL), which was introduced by Martius and Lampert [62] and extended with the division operation in the final layer by Sahoo et al. [72]. This method uses a neural network and replaces its activation functions with atomic units such as elementary functions. The neural network itself represents a very complex function. A special regularization scheme is used to find the sparse equation based on the data. This method is called *equation learning* and is the main topic of this thesis. More detailed information on the Equation Learner is provided in the following sections 3.2 – 3.5.

[62] Martius and Lampert (2016), "Extrapolation and learning equations"

[72] Sahoo, Lampert, and Martius (2018), "Learning Equations for Extrapolation and Control"

equation learning

Inverse Mapping of Data to Plausible Equations

The most recent and highly sophisticated and promising approach to tackle symbolic regression is to learn the inverse mapping of data to plausible equations directly. This is a rather new perspective, strongly influenced by deep learning and the concept of pre-training. In contrast to the previous approaches, which treated each dataset independently, this approach utilizes accumulated data from various instances of symbolic regression problems. Once trained, the pre-trained model can quickly solve specific symbolic regression cases. A first attempt to exploit this concept was the "Dreamcoder" by Ellis et al. [21]. The Dreamcoder aims to learn a probabilistic mapping of data to mathematical expressions. It has a simulation program to generate synthetic datasets. The mapping is learned by combining two kinds of domain knowledge. (1) Explicit declarative knowledge is built using a library of relevant equation building blocks and (2) implicit procedural knowledge is learned as a domain-specific search strategy by a neural network. With the help of these two kinds of expertise, the system can quickly suggest potential solutions for new datasets.

[21] Ellis, Wong, Nye, Sablé-Meyer, Morales, Hewitt, Cary, Solar-Lezama, and Tenenbaum (2021), "DreamCoder: bootstrapping inductive program synthesis with wake-sleep library learning"

A rigorous implementation to solve the inverse mapping using pre-trained models has been presented by Biggio et al. [3], called NeSymReS. The approach is motivated by previous achievements of large pre-trained neural networks such as [6, 18, 47] based on the transformer architecture and large datasets. By combining a set-transformer [50] as encoder and a regular transformer [82] as

[3] Biggio, Bendinelli, Neitz, Lucchi, and Parascandolo (2021), "Neural Symbolic Regression that scales"

Table 3.1: Comparative overview of various symbolic regression methods highlighting their features and constraints. This table is adapted from Camps-Valls et al. [9] and summarizes the embeddability, scalability, speed, restrictions, and the required domain knowledge for each method.

Method	Embeddable	Scaling	Speed	Restriction	Domain Knowledge
Genetic Programming [73]	×	×	Slow	For small systems	Elementary-functions, complexity of terms
FEYNMAN AI [80]	×	×	Slow	For physical systems in canonical form	Physics: units, symmetries
DSR [67]	×	×	Medium	Small input dim	Training domain
FFX [63], SINDY [7]	✓	✓*	Blazing	Needs known library	Training domain
EQL [72]	✓	✓	Slow	base functions limited, sometimes less concise	Elementary-functions, complexities
NESYMRES [3]	×	×	Fast	Small input dim	Training set

* It scales well to large output sizes; for high-dimensional input strong structural assumptions are required.

decoder, a pre-trained model could be learned from a very large set of synthetic equation-dataset pairs. It can process an entire dataset as input and directly outputs possible equation structures. The constant placeholders are then optimized with BFGS. Neither a search strategy nor a library of building blocks is necessary. Only a set of tokens, including input variables, elementary functions, mathematical operators, and constant placeholders, is required for this process.

3.1.1 Challenges in Symbolic Regression

This section discusses key aspects such as scaling, embeddability, speed, and domain knowledge that symbolic regression and its various methods encounter. A summary of the main aspects for the methods discussed previously is presented in table 3.1.

Scaling Symbolic regression is usually applied to small-scale problems. Here we discuss its applicability to more intricate systems that require larger or more complex equations. This includes the challenge of efficiently navigating the hypothesis space of equation structures, which grows exponentially with increasing equation size. Genetic algorithms are generally suitable for small-scale problems, where the target function is relatively simple. Although they are basically designed for exponentially growing hypothesis spaces, their effectiveness decreases when applied to complex systems. Methods such as FFX/SINDY require a comprehensive library of building blocks. Without additional assumptions and restrictions, the number of potential building blocks increases exponentially with system complexity, as these building blocks

are also derived combinatorially from atomic units. In contrast, the EQL method utilizes advances in machine learning, which enables effective scaling to a large number of parameters and large amounts of data. It uses gradient descent and can therefore process all equation components simultaneously.
In chapter 4 we extend the expressivity of the Equation Learner to atomic functions with singularities, such as logarithm or division, and introduce methods to stabilize training and improve sparsity inducing optimization [84].

[84] Werner, Junginger, Hennig, and Martius (2021), "Informed Equation Learning"

Embeddability It is intriguing to embed symbolic regression into larger computational frameworks or deep learning systems to address the lack of interpretability often associated with these models. Especially for deep learning systems, differentiability is crucial for end-to-end system training. Therefore, methods such as FFX/SINDy and EQL are particularly well suited for integration into more comprehensive frameworks. For example, Champion et al. [10], have shown how SINDy can be combined with a deep autoencoder to effectively learn both an effective coordinate system and the corresponding sparse governing equations of a system. Furthermore, the Equation Learner (EQL) model proves to be highly adaptable for integration into larger deep learning architectures. This adaptability is beneficial for applications such as the discovery of partial differential equations (PDEs) [53], or for integration with convolutional networks to extract and process digits for arithmetic tasks on MNIST [38]. In addition, Lin et al. [51] highlights the integration of EQL for modeling energy functions based on dynamic observations in the field of density functional theory.

[10] Champion, Lusch, Kutz, and Brunton (2019), "Data-driven discovery of coordinates and governing equations"

[53] Long, Lu, and Dong (2019), "PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network"

[38] Kim, Lu, Mukherjee, Gilbert, Jing, Čeperić, and Soljačić (2020), "Integration of neural network-based symbolic regression in deep learning for scientific discovery"

[51] Lin, Martius, and Oettel (2020), "Analytical classical density functionals from an equation learning network"

Speed The speed comparison in this paragraph refers to the performance per computing time and is based on the results of Biggio et al. [3]. This is a relevant quantity for the practitioner. Symbolic regression is an NP-hard problem [83]. We therefore refrain from comparing the time needed to find the global optimum. NeSymReS is very fast in finding suitable equations. The model can learn a suitable prior through pre-training and apply it during test time. Classical genetic programming and DSR are considerably slower, but approach the speed of NeSymReS with longer computing times. In contrast, EQL is expected to be the slowest method on small-scale problems due to extended training periods. SINDy/FFX is excluded from this comparison because it relies on a library of predefined building blocks. However, if this condition is met, these kinds of methods are typically the fastest.

[83] Virgolin and Pissis (2022), "Symbolic Regression is NP-hard"

Domain Knowledge Expert knowledge is essential for symbolic regression, as it is readily available across various domains of application. The selection of equation structures often depends on the domain, with certain structures being more plausible than others. As such, integrating domain knowledge into symbolic regression is a rapidly growing field of interest. This integration typically involves choosing relevant atomic units and tailoring their frequency or domain-specific complexity. Genetic algorithms offer the flexibility to select and modify these properties intuitively. More advanced types of domain knowledge might include specific functional forms and their derivatives [44] or axiomatic constraints [11]. In the FFX/SINDy framework, domain knowledge can be integrated by defining the building blocks within the library and adjusting the optimization through weighted sparsity regularization. NeSymReS allows for the customization of the training dataset to implement domain knowledge, though this method has not been extensively explored or developed.

[44] Kronberger, Franca, Burlacu, Haider, and Kommenda (2021), "Shape-Constrained Symbolic Regression—Improving Extrapolation with Prior Knowledge"

[11] Cornelio, Dash, Austel, Josephson, Goncalves, Clarkson, Megiddo, Khadir, and Horesh (2023), "Combining data and theory for derivable scientific discovery with AI-Descartes"

Incorporating expert knowledge into Equation Learning presents several challenges, notably maintaining differentiability for gradient-based methods and ensuring convergence in sparse optimization. In chapter 5, we propose methods for integrating expert knowledge effectively within the EQL framework. To this end, in section 4.3 we extend the expressivity of the EQL with a suitable relaxation for functions with divergences, which expands the selection of potential elementary functions available for domain-specific applications. Additionally, in section 5.2, we illustrate how the "copy units" introduced in section 4.4 serve to exclude domain-specific combinations of functions from the EQL architecture. Section 5.3 discusses how the application of a user-defined weighted L_0 regularization can influence the relative frequency of atomic unit utilization.

3.2 Neural Network Approach to Equation Learning

This thesis focuses on the EQL approach as a differentiable relaxation of symbolic regression, called *equation learning*. It is supposed to infer a parametric, differentiable function $\boldsymbol{f}_{\boldsymbol{\theta}}$, which maps d-dimensional input $\boldsymbol{x}$ to d'-dimensional output $\boldsymbol{y}$, represented as

$$\boldsymbol{y}_i = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + \epsilon \ . \tag{3.1}$$

The desired mathematical expressions are compositions of atomic units consisting of a set of atomic functions $\{f_i\}_{i \leq F}$ and a set of atomic operations that connect the single functions. The former can be, e.g., functions like {sin, sqrt, log, ...} and they can be connected by operations such as $\{\pm, *, /, \circ, ...\}$. The space of all possible equation structures given a set of atomic units is called *hypothesis space*. The complexity and size of this hypothesis space grows exponentially with each additional atomic unit introduced into an equation. Distinct from traditional symbolic regression, the structure of $\boldsymbol{f_\theta}$ and its parameters are inferred through sparsity inducing, gradient-based optimization within a neural network.

hypothesis space

Conventional symbolic regression frameworks typically presuppose the absence of noise in their datasets. However, in practical settings, the presence of noise ($\boldsymbol{\epsilon}$) is an inevitable aspect of data collection. To address this, the proposed framework incorporates an explicit noise term $\boldsymbol{\epsilon}$, which acknowledges the stochastic nature of real-world data. The dataset $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{Y})$ is assumed to be independent and identically distributed (iid.), and it contains $|\mathcal{D}|$ datapoints.

noisy measurements

Furthermore, from a Bayesian point of view, an accuracy measure can be derived from the noise distribution. For example, with isotropic Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$, with variance σ^2. The log-likelihood corresponds to an isotropic Gaussian distribution. Thus, the negative log-likelihood leads to the *mean squared error* (MSE) as an accuracy measure

$$\mathcal{L}_\mathcal{D} = \frac{1}{2\sigma^2 |\mathcal{D}|} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}} \|\boldsymbol{y}_i - \boldsymbol{f_\theta}(\boldsymbol{x}_i)\|_2^2 \,. \tag{3.2}$$

The following section examines how to construct such equation learning neural networks.

3.3 Equation Learning Neural Networks

This section introduces a generic method using deep neural networks, named *Equation Learner* EQL [62, 72], to tackle equation learning. They replace standard activation functions in a multi-layer feed-forward neural network with atomic units (sin, cos, identity, $*$) and a Lasso regularization [78]. In principle, this approach can be extended to all continuously differentiable functions with unbounded domains. The neural network itself represents the set of all considered equations of the hypothesis space within its architecture. During training irrelevant parts are omitted, leading to the convergence towards a sparse representation that represents the learned equation itself. This design allows seamless integration into larger neural network frameworks enabling full end-to-end training.

[62] Martius and Lampert (2016), “Extrapolation and learning equations”

[72] Sahoo, Lampert, and Martius (2018), “Learning Equations for Extrapolation and Control”

[78] Tibshirani (1996), “Regression shrinkage and selection via the lasso”

continuously differentiable functions

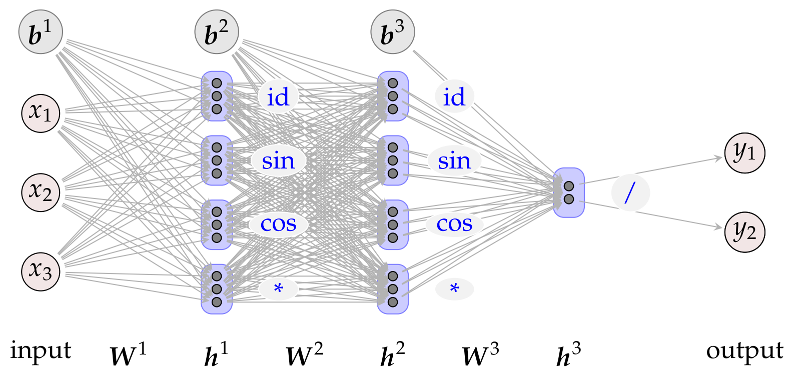


Figure 3.1: This figure shows the architecture of EQL$^{\div}$ with three hidden layers $\boldsymbol{h}$. It has a three-dimensional input $\mathbb{R}^3$ and a two-dimensional output $\mathbb{R}^2$. The first two layers apply trigonometric functions (sin, cos), muliplication ($*$) and the identity operation (id) with three nodes per atomic unit. The last hidden layer $\boldsymbol{h}^3$ applies two division units. This is a generic framework and its number of layers L, the number of atomic units as well as the choice of atomic units can be adjusted accordingly.

The architecture consists of L layers, which is also the maximum number of possible function compositions. Each hidden layer applies a non-linear transformation, which consists of u unary atomic units $f_{i\leq u} : \mathbb{R} \longrightarrow \mathbb{R}$ and v binary atomic units $g_{j\leq v} : \mathbb{R}^2 \longrightarrow \mathbb{R}$. The latter can be, e.g. division a/b or multiplication $a \cdot b$. These hidden layers can be similarly expressed as equation (2.8)

$$\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l \tag{3.3}$$

$$\boldsymbol{h}^l = (f_1(z_1^l), .., f_u(z_u^l), g_1(z_{u+1}, z_{u+2}), \dots). \tag{3.4}$$

identity units

The atomic unit id is of particular importance. It enables the network to pass on learned features from previous layers to intermediate layers. Without these units, each hidden layer adds a nesting to the equation. This property is discussed in more detail in the context of copy units in section 4.4. The EQL architecture by Martius and Lampert considered $L-1$ hidden layers with atomic units. The last layer L is a linear combination of all learned features (output = $\boldsymbol{z}^L = \boldsymbol{W}^L \boldsymbol{h}^{L-1} + \boldsymbol{b}^L$). and the input is represented by the zeroth layer (input = $\boldsymbol{h}^0$. This approach does not include common atomic units, such as division or logarithms, which are characterized by a half-bounded domain or by a singularity. This is a significant limitation of the original EQL framework. The addition of division units to the equation learning neural network introduced by Sahoo et al. [72] is a significant enhancement designed to model equations with division operations and is crucial for accurately representing physical systems that often include division units. The proposed architecture is called EQL$^{\div}$. It modifies the traditional EQL network structure by incorporating division units in the output layer. In this case the last layer L is given by division units

division units

$$\text{output} = \boldsymbol{h}^L = \left({}^{z_1^L}/_{z_2^L}, {}^{z_1^L}/_{z_2^L}, \dots \right) . \tag{3.5}$$

The proposed algorithm uses a hard-coded curriculum, which depends on the number of epochs itself outlined in equation (3.7). This poses a significant challenge when adapting to new datasets with different sample sizes from different systems. Figure 3.1 shows a possible architecture of the EQL$^{\div}$ with three hidden layers. The network has a three-dimensional input $\mathbb{R}^3$ and a two-dimensional output $\mathbb{R}^2$. The first two layers apply trigonometric functions (sin, cos), multiplication $(*)$ and the identity operation (id) with three nodes per atomic unit. The last hidden layer $\boldsymbol{h}^3$ applies two division units to map onto the correct output dimension. Identity mappings enable the network to learn equations that posses fewer nonlinear units than the network's overall depth. As discussed, the number of layers L, the number of atomic units and the choice of atomic units can be adjusted according to the dataset.

3.4 Training Equation Learning Neural Networks

Continuous, gradient based optimization of an equation learning neural network with division units is not trivial. It can break down due to two primary issues: First, the cascading transformation of intermediate results within a deep architecture may map values outside of the restricted input domains[1]. Second, singularities present in the atomic units or their derivative can induce excessively large gradient values, thereby rendering optimization highly unstable.

1: Applying domain-limiting mapping functions, such as softplus, may seam feasible to address this issue, but it is suboptimal since it can significantly alter the final symbolic equation.

This section gives a brief introduction of the necessary methods to train EQL$^{\div}$ with division units introduced by Sahoo et al. For a detailed discussion please read "Learning Equations for Extrapolation and Control" [72].

[72] Sahoo, Lampert, and Martius (2018), "Learning Equations for Extrapolation and Control"

3.4.1 Regularized Division

The architecture's primary challenge is handling potential discontinuities and infinite values when the divisor approaches zero. Therefor a regularized division-activation function $\text{div}_\phi(a, b)$ is defined as

$$\text{div}_\phi(a,b) := \begin{cases} \frac{a}{b} & \text{if } b > \phi \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The parameter $\phi > 0$ is a predefined threshold that progressively relaxes the regularization of the division operation during training. If the denominator is smaller than the threshold value, the gradient is also zero. Initially, a high threshold ϕ is used to avoid steep gradients and to ensure numerical stability. As training progresses, ϕ is gradually decreased according to a predefined curriculum

curriculum

$$\phi(t) = \frac{1}{\sqrt{t+1}} \,. \tag{3.7}$$

This approach allows the network to start with a simpler, more regularized version of division, and gradually moves towards the true division operation as it learns the underlying structure of the data. This curriculum facilitates stable learning and helps the network to converge to meaningful solutions that can generalize well beyond the training set.

3.4.2 Penalty Term

To discourage the network from producing negative or very small denominators, a domain penalty term is introduced. This term penalizes prohibited inputs to each division unit, which diverts the optimization process from undesirable regions. The penalty term for a denominator b is defined as

$$p_\phi(b) := \max(\phi - b, 0) \ . \tag{3.8}$$

All denominator values (b) that are smaller than ϕ are penalized. Aggregating these penalties across all division units in the network forms a global domain penalty term $\mathcal{L}_\phi$.

3.4.3 Penalty Epochs

Penalty epochs are a specific training phase that is introduced at regular intervals to further enforce constraints on the network's output in the final test domain. During a penalty epoch, N input datapoints are randomly sampled within the anticipated test range[2], but are not associated with output labels. To ensure that the network's predictions do not deviate significantly from observed magnitudes, a *bound penalty* term $\mathcal{L}_{\text{bound}}$ is introduced. It penalizes overly large output values of the network, thus reduces the risk of overfitting to the training data[3]

2: the test range can include unseen extrapolation regions

bound penalty

3: e.g. learning a polynomial of too high degree

$$\mathcal{L}_{\text{bound}} = \sum_{i=1}^{N} \| \max(|\boldsymbol{f}_{\boldsymbol{\theta}}(x_i)| - B, 0) \|_1 \ . \tag{3.9}$$

The upper limit of acceptable output magnitudes B is typically set based on the observed data. All predicted values that are outside of the d' dimensional cube of range B are penalized. The network is then trained using a specific penalty loss term that is a superposition of the penalty term $\mathcal{L}_\phi$ to avoid generating negative denominators in the test domain and the bound penalty term $\mathcal{L}_{\text{bound}}$ given by

$$\mathcal{L}_{\text{penalty}} = \delta \mathcal{L}_\phi + \eta \mathcal{L}_{\text{bound}} \ . \tag{3.10}$$

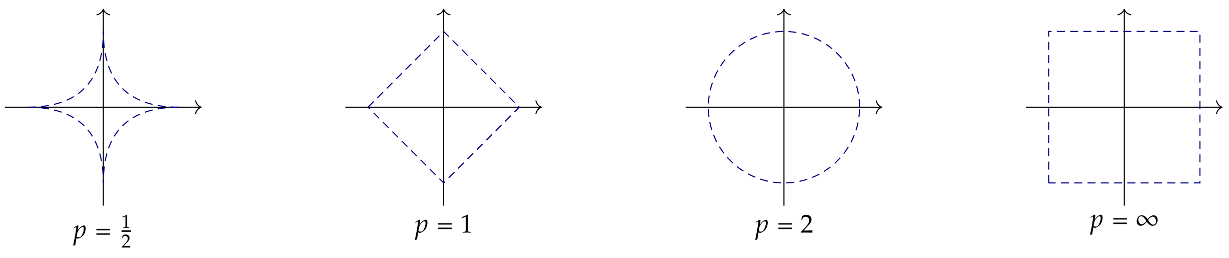


Figure 3.2: The figure illustrates contours of the regularization term given in equation (3.11), for the parameter values $p \in \{1/2, 1, 2, \infty\}$ from left to right.

The weighting parameters δ, η are hyperparameters that can be adjusted accordingly to the dataset.

3.4.4 Regression with Sparsity Regularization

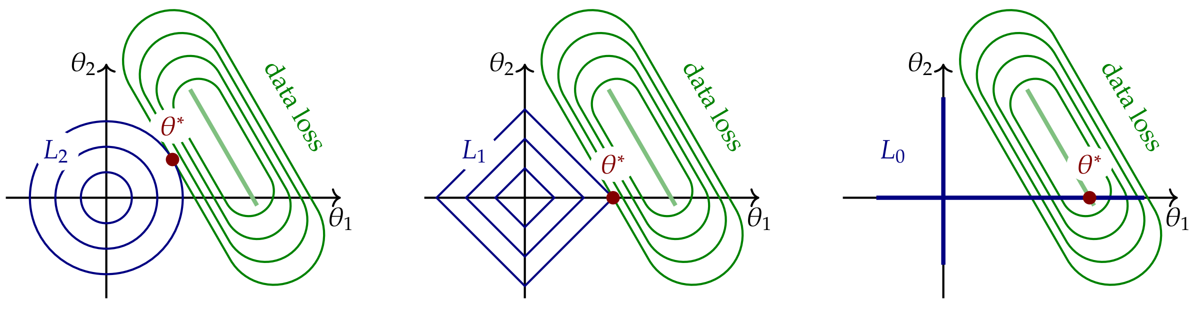
The goal of the equation learning neural network is to achieve a sparse representation that is the desired, simple and accurate equation. Common regularization techniques employed to attain sparsity include L_2 (ridge), L_1 (lasso) and L_0 regularization given by

$$\mathcal{L}_{\mathrm{C}}^{p} = \|\boldsymbol{\theta}\|_p^p = \sum_{i=1}^{|\boldsymbol{\theta}|} |\theta_i|^p \,. \tag{3.11}$$

Here, $\mathcal{L}_{\mathrm{C}}$, refers to complexity loss function. The total number of parameters in the network is given by $|\boldsymbol{\theta}|$. Figure 3.2 illustrates contours for different parameters p for equation (3.11). The L_0 regularization is particularly notable, as it provides a direct measure of model complexity by counting the non-zero parameters. It is not directly applicable in gradient-based optimization since it is not differentiable. This issue is discussed in section 4.5. A simple sketch outlined in figure 3.3 illustrates a mean squared error optimization under L_2, L_1 and L_0 regularization. Readers can refer to the work by Bishop ([4], chapter 3) for more information on regularized least squares. It emerges that the L_2 regularization leads to overall small parameters, but does not induce sparsity. Conversely, L_1 regularization encourages sparsity, which can be particularly beneficial for model interpretability and efficiency of the retrieved equation. Nevertheless, the sparse lasso optimization results in a consistent underestimation of the parameter values due to its inherent compromise between minimizing the accuracy loss and the regularization term. The equation for regression with a mean squared error as data loss $\mathcal{L}_{\mathcal{D}}$ and sparsity regularization $\mathcal{L}_{\mathrm{C}}^{p}$ is expressed as

[4] Bishop (2006), "Pattern recognition and machine learning"

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} \|\boldsymbol{y}_i - \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2 + \lambda \|\boldsymbol{\theta}\|_p^p \,. \tag{3.12}$$



(a) L_2 regularization (b) L_1 regularization (c) L_0 regularization

Figure 3.3: The figure illustrates the contours of the sparsity inducing loss function given in equation (3.12) for three different sparsity regularizations $\{L_2, L_1, L_0\}$ from left to right. It is a linear combination of the quadratic data loss and a sparsity regularization L_p.

Contour plots for different sparsity regularizations are shown in figure 3.3. In this equation λ represents the regularization strength, balancing the trade-off between the model's accuracy and its complexity. Larger values of λ prioritize sparsity at the potential expense of model accuracy. This balance is crucial for achieving optimal solutions that are both accurate and simple and facilitate interpretation and generalization.

3.4.5 Trainingphases

The objective to train an equation learning neural network is split in three relevant loss components, namely data loss $\mathcal{L}_\mathcal{D}$, complexity loss $\mathcal{L}_\mathrm{C}$ and penalty loss $\mathcal{L}_\phi$ given by

$$\mathcal{L} = \mathcal{L}_\mathcal{D} + \lambda\, \mathcal{L}_\mathrm{C} + \delta\, \mathcal{L}_\phi \,. \tag{3.13}$$

The strength of the complexity loss λ is an important parameter to control the sparsity of the retrieved equations. The weighting of the penalty term δ depends on the network's architecture itself and is an important hyperparameter to reduce instabilities related to the division units.

The training procedure is split in three training phases as proposed by Martius and Lampert [62] to address the challenges in sparse optimization of equation learning networks like a consistent underestimation of the parameter values and local minima.

initial training phase

In the *initial training phase* ($t < t_1$), complexity regularization is not applied ($\lambda = 0$), so that the network parameters can adjust freely and settle at appropriate initial values. This lack of constraints is essential to avoid early convergence towards suboptimal solutions.

intermediate training phase

This is followed by a *intermediate training phase* ($t_1 < t < t_2$) with regularization ($\lambda > 0$) to promote sparse network structures.

This step is crucial to reduce the risk of overfitting and lead the model to simpler, more interpretable solutions while preserving the necessary details.

In the *final training phase* ($t > t_2$), the focus shifts to optimizing the identified sparse representation, but without regularization ($\lambda = 0$). This approach addresses the challenges of underestimated parameter values and minor fluctuations of parameter values around zero. It explicitly masks all parameters with negligible magnitude to zero[4] and excludes them from further optimization in subsequent epochs. This decision is designed to maintain the network's focus on the most relevant features identified during the previous phase, thereby reinforcing a sparse yet effective representation.

final training phase

4: for instance, setting $\theta = 0$ if $|\theta| < 0.001$

3.5 Model Selection Criteria

In machine learning, models are typically selected based on the validation error that is evaluated on an interpolation dataset $\mathcal{D}^{\text{valid}}_{\text{int}}$ from the same data domain as the training dataset $\mathcal{D}^{\text{train}}$. The *interpolation error* for the root mean squared error RMSE is thus given by

interpolation error

$$\nu_{\text{int}}\left[f_{\boldsymbol{\theta}}\right] = \sqrt{\frac{1}{M} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}^{\text{valid}}_{\text{int}}} \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|^2}\ , \ M = |\mathcal{D}^{\text{valid}}_{\text{int}}| \tag{3.14}$$

It has the draw back, that while effective in fitting data, it can lead to overfitted models that are difficult to interpret and generalize poorly to new data. This selection criterion V_{int} is defined as follows

$$\mathrm{V}_{\text{int}} = \underset{f^*_{\boldsymbol{\theta}} \in \{f_{\boldsymbol{\theta}}\}}{\arg\min} \left[\nu_{\text{int}}[f^*_{\boldsymbol{\theta}}]\right] \ . \tag{3.15}$$

In the context of equation learning the model selection must ensure both accuracy and generalizability of the equation. Especially when extending the prediction to unseen areas, the equation must extrapolate reliably. Traditionally, in equation learning, models are selected based on a balance between validation error and model complexity, guided by the principle of Occam's razor. Where optimal models maintain simplicity while ensuring an adequate accuracy. This leads to Pareto optimal solutions. They form the Pareto front that is a central tool in understanding the trade-offs between different objectives, here accuracy and simplicity. Simplicity is commonly quantified by the number of active parameters $s[f_{\boldsymbol{\theta}}] = \|\boldsymbol{\theta}\|_0$. In this context, the simplicity is described by the L_0 regularization[5]. A model's position on the Pareto front indicates that no other model can outperform it in one objective without

5: The interpretation of complexity in the field of equation learning is not well-defined; it ranges from the simple counting of atomic units to the application of more sophisticated complexity measures.

degrading performance in another. This embodies the essence of multi-objective optimization. Sahoo et al. [72] propose a novel selection criterion $\mathrm{V_{int}}$-S based on a normalized interpolation validation error $\tilde{\nu}_{\mathrm{int}}[f_{\boldsymbol{\theta}}]$ and a normalized sparsity measure $\tilde{s}[f_{\boldsymbol{\theta}}]$, given by

$$\mathrm{V_{int}\text{-}S} = \underset{f^*_{\boldsymbol{\theta}} \in \{f_{\boldsymbol{\theta}}\}}{\arg\min} \left[\alpha\, \tilde{\nu}^2_{\mathrm{int}}[f^*_{\boldsymbol{\theta}}] + \beta\, \tilde{s}^2[f^*_{\boldsymbol{\theta}}]\right] , \tag{3.16}$$

where α and β are coefficients that weight the relative importance of accuracy and simplicity[6]. Here, both validation error ν_{int} and sparsity measure s are normalized to $[0, 1]$ across all found equations $\{f_{\boldsymbol{\theta}}\}$.

6: Sahoo et al. [72] demonstrate that equal coefficients $\alpha = \beta$ yield favorable results.

extrapolation error

In the event that some extrapolation datapoints $\mathcal{D}^{\mathrm{valid}}_{\mathrm{ex}}$ are accessible, the *extrapolation error* can be calculated, defined here as an example for the RMSE as

$$\nu_{\mathrm{ex}}\left[f_{\boldsymbol{\theta}}\right] = \sqrt{\frac{1}{M} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}^{\mathrm{valid}}_{\mathrm{ex}}} \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|^2} \;, \; M = |\mathcal{D}^{\mathrm{valid}}_{\mathrm{ex}}| \,. \tag{3.17}$$

Sahoo et al. [72] empirically show that the complexity measure in equation (3.16) can be replaced by a normalized extrapolation error due to the additional data from the extrapolation domain

$$\mathrm{V_{int\&ex}} = \underset{f^*_{\boldsymbol{\theta}} \in \{f_{\boldsymbol{\theta}}\}}{\arg\min} \left[\alpha\, \tilde{\nu}^2_{\mathrm{int}}[f^*_{\boldsymbol{\theta}}] + \gamma\, \tilde{\nu}^2_{\mathrm{ex}}[f^*_{\boldsymbol{\theta}}]\right] . \tag{3.18}$$

In this framework, the coefficients α and γ balance the significance of accuracy within the domains of interpolation and extrapolation[7]. This approach ensures that the selected models are both simple and accurate in the domain of interpolation, while maintaining their predictive capabilities in the area of extrapolation.

7: Sahoo et al. [72] demonstrate that equal values for the parameters $\alpha = \gamma$ yield favorable results.

To obtain a Pareto front that reflects the spectrum of optimal equations representing the trade-off between complexity and accuracy, Martius and Lampert [62] recommends training equation learning neural networks by changing the strength of regularization and the number of hidden layers.

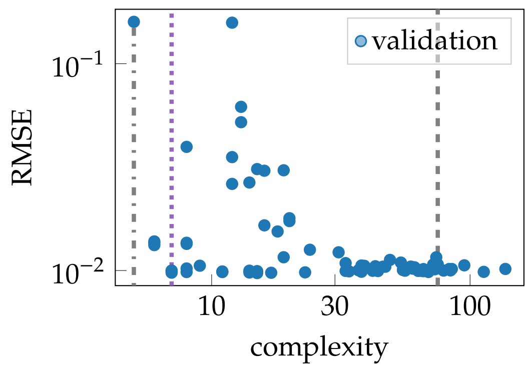
example selection process

To provide a concrete understanding, an example of an equation selection process with the $\mathrm{V_{int\&ex}}$ selection criterion is shown in figure 3.4. Figure 3.4(a) shows the RMSE in relation to the complexity of different equations. The groundtruth equation is

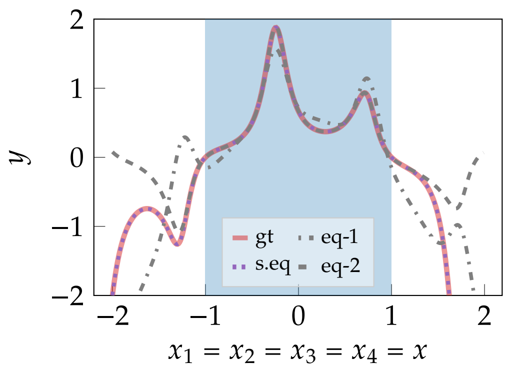
$$y = (1 - x_2^2)/(\sin(2\pi\, x_1) + 1.5) \,. \tag{3.19}$$

The Pareto front in figure 3.4(a) shows an optimal validation error of 0.01, which corresponds to the noise level of the data[8]. To underscore the importance of the equation selection process,

8: The training dataset consists of 10^4 randomly sampled datapoints from $[-1, 1]^4$. For the validation dataset 10% of the training dataset is used. The datapoints are corrupted with standard normal noise of standard deviation 0.01.



(a) pareto front



(b) equations

Figure 3.4: The left panel shows a pareto front of equations. Each point corresponds to an independent run of the iEQL with a different complexity regularization. The right panel displays a slice through the input space of equation (3.19). The blue shading indicates the train domain $[-1, 1]^4$. The thick line is the ground truth (gt) given by equation (3.19). The selected equation (dotted line, s.eq) was determined with the $V_{\text{int\&ex}}$ selection criterion. Additionally, a too simple (dashdotted line, eq-1) and a too complex equation (dashed line, eq-2) are plotted for comparison.

the selected equation is compared with less complex and more complex alternatives on the test domain $[-2, 2]^4$ in figure 3.4(b). It demonstrates the potential pitfalls of improper model selection: overly simplistic equations are tempting for their clarity. However, they may fall short in validation and extrapolation performance. Conversely, too complex equations might align with the noise level of the validation data, but fail to extrapolate. In this example, the selection criterion $V_{\text{int\&ex}}$ successfully identified an equation with sufficient complexity that extrapolates. It highlights the critical balance between complexity of the equation and the accuracy of the predictions.

Part II

Informed Equation Learning

Enhancing Expressivity and Training Stability of the Equation Learner 4

A large expressivity of an equation learner is desirable to apply it to a broad variety of datasets and scale it to real-world applications. This chapter addresses the research question *"How to broaden the expressivity of equation learning neural networks and train them efficiently?"*. Section 4.2 provides a comprehensive overview of related literature. In section 4.3, we introduce a robust framework to train equation learning neural networks with atomic units that exhibit singularities or operate on a restricted domain. Section 4.4 proposes *copy units*, an approach to avoid scanning different numbers of hidden layers when retrieving the Pareto front. Finally, section 4.5 discusses the adoption of an L_0 regularization scheme that does not cause parameter shrinkage during the training process. Section 4.6 provides the conclusion of the methods introduced and connects them to the subsequent chapter, which explores the integration of domain and expert knowledge within our framework of equation learning.

4.1 Motivation

atomic units with singularities

The equation learner (EQL) is well-suited for handling large-scale problems. However, it faces challenges when dealing with equations that involve standard mathematical operations, such as logarithms, square roots or divisions. The singularities associated with these operations induce instabilities during training due to unbounded gradient values. These instabilities are further compounded in deep architectures, as cascading transformations can render certain functions, such as logarithms and square roots, inapplicable due to their restricted domains, which can lead to training collapse. Moreover, successful training on the training domain does not guarantee the avoidance of singularities in relevant extrapolation domains. Singularities may only be shifted to regions that are not covered by the training data. A common method in deep learning to solve the first issue is to applying domain-limiting mapping functions, such as softplus[1]. This is not a suitable solution for equation learning, since it compromises the final symbolic equation with additional expressions. It enforces to use singular functions alongside the domain-limiting mapping function[2].

1: softplus = $\log(1 + e^x)$

2: e.g. $\log(\text{softplus}(x))$

Previous attempts to integrate division into the equation learner framework, such as the EQL$^{\div}$ by Sahoo et al. [72], have been limited to including division units only in the final layer and require a predefined training sequence. In contrast, our work introduces a comprehensive and robust training method for handling atomic units with singularities in all layers. This approach not only takes into account the constraints associated with different datasets, but also replaces the predefined training sequence with a learnable parameter to facilitate the application.

copy units

In order to obtain a Pareto front that reflects the spectrum of optimal equations, it is common to train multiple equation learning neural networks with differing numbers of hidden layers. Each network configuration typically requires a unique number of training epochs to ensure convergence. Section 4.4 introduces the concept of *copy units*, which simplifies this training process. Copy units allow for the simultaneous training of various network architectures using the same number of epochs and removing the need for extensive grid searches over the number of hidden layers. This method favors simpler and more efficient network architectures.

L_0 regularization

The traditional EQL architecture suffers from parameter shrinkage due to the L_1 sparsity regularization. To address this challenge, an additional final training phase is necessary without regularization and a masked structure to avoid the shrinkage. In section 4.5 we propose to use L_0 regularization, which does not shrink the weights during training and allows to naturally align a domain specific complexity measure with the regularization. This approach differs from the usual L_1 regularization in that no final training phase of the identified sparse representation is necessary. However, a probabilistic optimization scheme is required for this approach.

4.2 Related Work

This section summarizes previous research and developments.

[19] Dubčáková (2011), "Eureqa: software review"
[73] Schmidt and Lipson (2009), "Distilling Free-Form Natural Laws from Experimental Data"
[45] Kusner, Paige, and Hernández-Lobato (2017), "Grammar variational autoencoder"
[47] Lample and Charton (2020), "Deep Learning For Symbolic Mathematics"

[3] Biggio, Bendinelli, Neitz, Lucchi, and Parascandolo (2021), "Neural Symbolic Regression that scales"

Equation learning is commonly addressed in the context of symbolic regression with genetic programming and evolutionary algorithms [41, 48, 64, 75]. A major success was the automated discovery of natural laws by Schmidt and Lipson [19, 73]. Such transparent models are becoming increasingly important as a complement to the predominant black-box models in the field of machine learning Zaremba et.al. [89] show how to apply n-gram models and recurrent neural networks to guide a tree search for efficient mathematical identities. Kusner et.al. [45] use Bayesian optimization to search for equations in the latent space of a variational autoencoder with a prior on mathematical constraints. Lample et.al. [47] use seq2seq transformers to solve mathematical integration and ODEs. Biggio et.al. [3] apply set-transformers trained on an unbounded set of equation-dataset pairs to guide the search. Inspired by Physics, Udrescu et.al. [80, 81] exploit symmetries and separability in the dataset to enhance the search. Other approaches use gradient information about the learned expression during training. This was addressed in a reinforcement learning formulation by Petersen et.al. [67] via risk-seeking policy gradients. A different, yet powerful approach that exploits gradient information are equation learning neural networks (EQL) first introduced by Georg Martius et.al. [62] and extended to division by Sahoo et.al. [72]. They represent a complex equation within their architecture, with different kinds of activation functions (e.g. $\{\cos, \sin, *, /, \dots\}$) in each hidden layer. During training, irrelevant parts are omitted and neural network converges to a sparse representation that is the wanted equation itself. Its design allows to integrate the EQL within larger neural network architectures and train them end-to-end. Kim et.al. [38] integrate it within other deep learning frameworks for scientific discovery. Long et.al. [53] apply it to solve differential equations and Lin et.al. [51] obtain analytical expressions of classical free energy functionals.

[80] Udrescu, Tan, Feng, Neto, Wu, and Tegmark (2020), "AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity"
[81] Udrescu and Tegmark (2020), "AI Feynman: A physics-inspired method for symbolic regression"

[67] Petersen, Larma, Mundhenk, Santiago, Kim, and Kim (2021), "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients"

[62] Martius and Lampert (2016), "Extrapolation and learning equations"

[72] Sahoo, Lampert, and Martius (2018), "Learning Equations for Extrapolation and Control"
[51] Lin, Martius, and Oettel (2020), "Analytical classical density functionals from an equation learning network"

4.3 Atomic Functions with Singularities

We present a robust framework designed to train equation learning neural networks with atomic units that have singularities or operate within a half-bounded domain. Our algorithm is designed for atomic function with singularities $f : \mathbb{D} \to \mathbb{R}$. Typically, such singular functions have support on a half-bounded domain $\mathbb{D} \equiv (a, \infty)$ exhibiting a singularity at a[3]. It also applies to division c/d under the assumption that real systems do not diverge in the domain of application. It is sufficient to only consider the positive branch of the hyperbola $1/d$ and let c choose the sign.

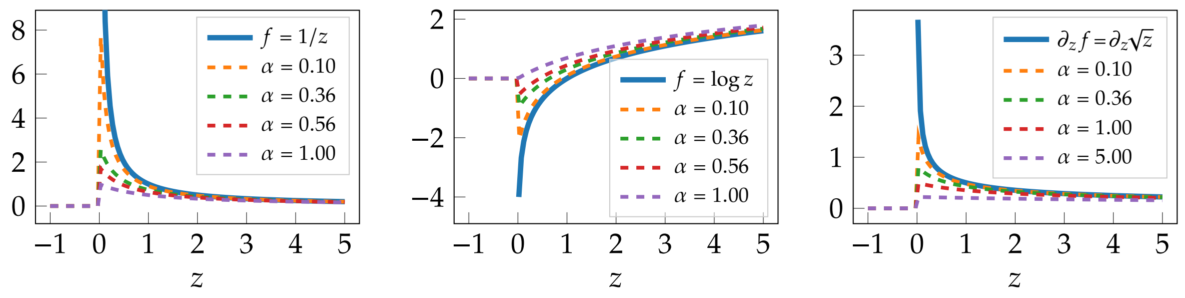
3: e.g. $\log : (0, \infty) \to \mathbb{R}$ with a singularity at $a = 0$

4.3.1 Learnable Relaxation

Cascading transformations of intermediate results in the deep architecture of the EQL can project values outside of the constrained input domains of singular functions. Therefore, the domain $\mathbb{D}$ of singular functions is continuously extended to $\mathbb{R}$ to avoid forbidden inputs during training. Formally, we redefine the singular function $f(z)$ as follows

$$\hat{f}(z) = \begin{cases} f(z), & \text{for } z > a, \\ 0, & \text{for } z \le a \end{cases} . \tag{4.1}$$

This reformulation extends equation (3.6) to functions with singularities. Still, training can be corrupted by unconstrained values



(a) $f(z) = 1/z$ (b) $f(z) = \log z$ (c) $f(z) = \partial_z \sqrt{z}$

Figure 4.1: The left figure (a) shows the relaxation of the division operation. The middle figure (b) shows the relaxation of the logarithm function and the left figure (c) shows the relaxation of the derivative of the squareroot function for different values of α from equation (4.2).

and exploding gradients in the neighborhood of the singularities. Therefore, we propose a learnable relaxation of the singular units that does not compromise the final symbolic equation. To counteract these issues, we propose a *learnable relaxation* of the singular functions that does not compromise the final symbolic equation. It is shown in figure 4.1 for the logarithm, division and the derivative of the square root function. It involves a shift in the input $\hat{z} = z + \alpha$, where $\alpha > 0$ is a learnable positive relaxation-parameter[4] that can be optimized alongside all other network parameters using gradient-based methods. Thus, the function with the proposed relaxation is

4: The choice of reparametrization is not unique. Choices like $\alpha = \log(1 + e^{\hat{\alpha}})$ or $\alpha = e^{\hat{\alpha}}$ are possible.

$$\hat{f}(\hat{z}) = \begin{cases} f(z + \alpha), & \text{for } z > a, \\ 0, & \text{for } z \leq a \end{cases} . \tag{4.2}$$

This adjustment ensures that the affine transformation, $z = W\,h + b$, incorporates the relaxation parameter α within its bias b without altering the fundamental structure of the final equation. This method assures a bounded maximum $C \geq 0$ of the absolute function and gradient values in the neighborhoodod of the singularity. It significantly reduces the risk of training instability and in contrast to Sahoo et al. [72] does not rely on a predefined threshold ϕ.

4.3.2 Domain Penalty

During training, an additional domain penalty loss $\mathcal{L}_{\text{sf}}$ is necessary to constrain the solution space of the EQL to networks that respect the domain $\mathbb{D}$ of all its singular functions $\{f_u(z_u)\} := \mathcal{SF}$, within the network.
For each singular function a structured penalty term ensures that during training, the network configurations remain within the bounds of the function domain, thereby avoiding inputs that could lead to undefined behavior in singular functions. It aligns with

equation (3.8) and is given by

$$\mathcal{L}^{\mathrm{u}}_{\mathrm{sf}}(z^u_i) = \begin{cases} 0, & \text{for } z^u_i > a\,, \\ |a - z^u_i|, & \text{for } z^u_i \leq a\,. \end{cases}$$

The domain penalty loss is the sum over all structured penalty terms and given by

$$\mathcal{L}_{\mathrm{sf}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{u \in \mathcal{SF}} \mathcal{L}^{\mathrm{u}}_{\mathrm{sf}}(z^u_i)\,. \tag{4.3}$$

The number of datapoints is denoted by N.

4.3.3 Penalty Epochs

In order to ensure that the domain constraints for singular functions also apply in the extrapolation domain and that the order of magnitude does not change drastically, intrinsic penalty epochs are necessary as proposed by Sahoo et.al. [72] and discussed in section 3.4.3. Penalty epochs consist of two contributions. The domain penalty loss $\mathcal{L}_{\mathrm{sf}}$ assures that the domain constraints are not violated within the test domain. The second contribution $\mathcal{L}_{\mathrm{bound}}$ (see equation (3.9)) assures that the order of magnitude does not change drastically on the test domain. During an intrinsic penalty epoch, N_p datapoints from the expected test domain are randomly sampled. These samples are not linked to any output labels. The loss function for the intrinsic penalty epoch is therefore defined as

$$\mathcal{L}_{\mathrm{penalty}} = \delta \mathcal{L}_{\mathrm{sf}} + \eta\, \mathcal{L}_{\mathrm{bound}}\,. \tag{4.4}$$

The weighting parameters δ, η determine the ratio of domain penalty $\mathcal{L}_{\mathrm{sf}}$ to bound penalty $\mathcal{L}_{\mathrm{bound}}$.

4.3.4 Objective Function for Training

The main objective function used for regularized training includes data loss $\mathcal{L}_{\mathcal{D}}$, complexity loss $\mathcal{L}_{\mathrm{C}}$, and domain penalty $\mathcal{L}_{\mathrm{sf}}$, each weighted by their respective coefficients, δ for the domain penalty and λ for the complexity

$$\mathcal{L} = \mathcal{L}_{\mathcal{D}} + \delta\, \mathcal{L}_{\mathrm{sf}} + \lambda\, \mathcal{L}_{\mathrm{C}}\,. \tag{4.5}$$

In this formulation, the magnitude of the complexity loss λ is important to control the sparsity level of the resulting equation. The strength of the domain penalty, influenced by the network's design, is an essential hyperparameter to minimize instability during training.

4.4 Feature Reuse with Copy Units

Equation learning neural networks require identity units to pass features from previous to intermediate layers. Without them, the network is limited to nested equations, each nesting adding an atomic unit. To obtain a representative Pareto front, many networks with different numbers of hidden layers are typically trained. We propose copying units that simplify training by enabling the simultaneous training of different network architectures.

Copy units allow an affine transformation to be applied to the outputs of all previous layers. This also applies to the inputs $\boldsymbol{x} = \boldsymbol{h}^0$. Mathematically, this can be represented as follows

$$\boldsymbol{z}^l = \boldsymbol{W}^l \tilde{\boldsymbol{h}}^{l-1} + \boldsymbol{b}^l \text{ , with: } \quad \tilde{\boldsymbol{h}}^{l-1} = (\boldsymbol{h}^0, \dots, \boldsymbol{h}^{l-1}) \tag{4.6}$$

$$\boldsymbol{h}^l = (f_1(z_1^l), .., f_u(z_u^l), g_1(z_{u+1}, z_{u+2}), \dots). \tag{4.7}$$

This implies that the outputs from each previous layer, denoted as $\boldsymbol{h}^{l<l'}$, are reused as inputs for every subsequent layer l', which is visually represented in figure 4.2 for a simple architecture. This approach draws inspiration from an existing concept that incorporated skip connections into densely connected convolutional networks to strengthen the propagation and reuse of features, as proposed by Huang et.al. [33]. Those skip connections allow identified sub-solutions (terms in the equation) to be available with no additional cost to all subsequent layers.

[33] Huang, Liu, Maaten, and Weinberger (2016), "Densely Connected Convolutional Networks"

The following example studies the characteristics of identity units

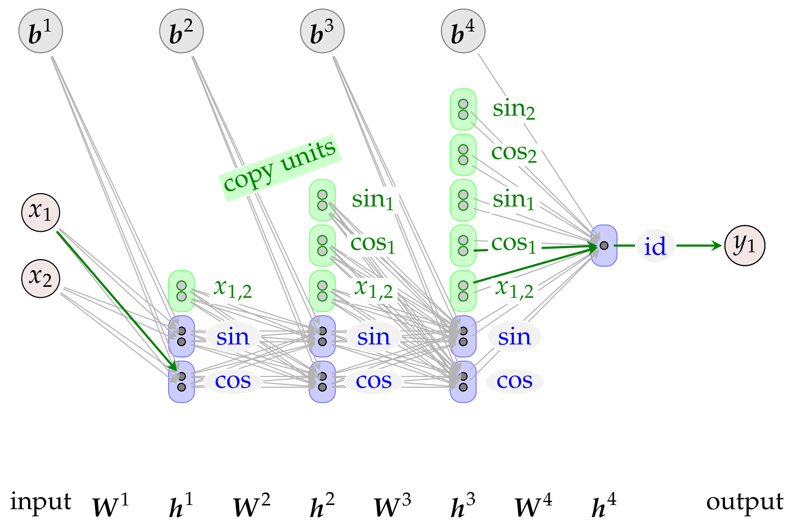


Figure 4.2: Simple equation learning neural network architecture with copy units in green. The green edges highlight a minimal representation of equation (4.8) with copy units and four hidden layers.

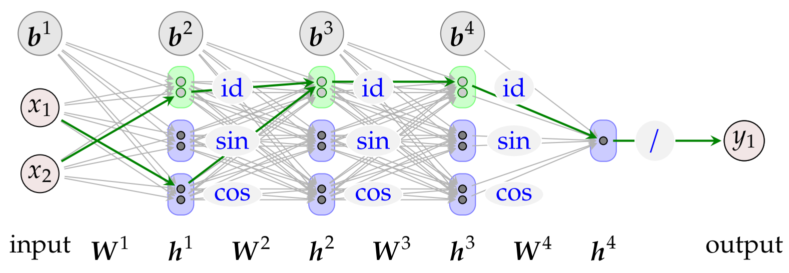


Figure 4.3: Simple equation learning neural network architecture with identity units in green. The green edges highlight a minimal representation of equation (4.8) with identity units and four hidden layers.

and copy units in neural networks using the equation

$$y = a_1 \cos(a_2 x_1) + a_3 x_2 \,, \tag{4.8}$$

with coefficients a_i. Identity units accumulate all features of the previous hidden layer in a linear combination and forward this combined input to the next layer. The number of necessary weights to represent a certain equation depends on the number of hidden layers. Traditional EQL-architectures with identity units as described in section 3.3 with L hidden layers must identify at least $(2 + L)$ coefficients to represent equation (4.8). For any unused hidden layer an additional connection is required, since the architecture does not support to skip hidden layers as illustrated in figure 4.3 for a simple architecture with four hidden layers. In this case the identified equation has six coefficients. Even with the simplest architecture ($L = 2$) the EQL requires more constants than the original equation.

In contrast, copy units ensure by design that the required minimum number of coefficients is independent of the number of hidden layers, provided that the architecture can represent the equation. Consequently, it eliminates the need for a grid search across different numbers of hidden layers. Therefore, figure 4.2 illustrates that equation (4.8) can be modeled using just three coefficients, as intended. Naturally, the copy units favor less nested expressions and allow unnecessary layers to be ignored. Moreover, the search for appropriate hyperparameters is simplified as the initial network can be chosen larger and more complex.

Copy units are of particular interest in section 5.2. They are essential for an architecture that allows combinations of atomic units to be prohibited, a feature that cannot be achieved with identity units.

4.5 Avoiding Parameter Shrinkage Through L_0 Regularization

An equation learner seeks sparse[5] and accurate representations of its internal structure for equation. Current equation learning frameworks like EQL or EQL$^{\div}$ suffer from parameter shrinkage due to the lasso sparsity regularization. To avoid this issue, we propose to use L_0 regularization, which does not shrink the weights during training, like L_1 regularization. The corresponding regularization loss is provided by

5: For equation learning neural networks, sparsity is only sought for the weight matrices rather than the biases, as this only involves a shift in the respective feature input and does not directly contribute to the complexity of the equation. In addition, in certain systems, biases are related to the calibration of measurement instruments.

$$\mathcal{L}_{\mathrm{C}}^{0} = \|\mathcal{W}\|_0 = \sum_{i \leq W} |w_i|_0 \quad \text{, with} \quad |w_i|_0 = \mathbb{I}[w_i \neq 0]\,. \tag{4.9}$$

The set of all weight matrices is defined as $\mathcal{W} = \{\boldsymbol{W}^l\}_{l \leq L}$. The total count of weight parameters within the network is denoted by $W := \sum_{l \leq L} |\boldsymbol{W}^l|$. The L_0-norm quantifies the count of non-zero weights. However, due to its lack of differentiability, it is incompatible with conventional gradient-based optimization methods. To address this, we propose to utilize a stochastic gradient-based optimization method for objectives with L_0 regularization. It is based on the findings of Louizos et.al. [55]. Minor adjustments are necessary to adapt the original algorithm to equation learning neural networks that require sparsity in the weights rather than the nodes. The objective loss function is given by

[55] Louizos, Welling, and Kingma (2017), "Learning Sparse Neural Networks through L_0 Regularization"

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2 + \lambda \|\mathcal{W}\|_0\,. \tag{4.10}$$

Specific choices of the scale parameter λ refer to well-known model selection criteria as the Bayesian Information Criterion (BIC, [74]) and the Akaike Information Criterion (AIC, [1]). Louizos et.al. suggest a set of non-negative stochastic gates to collectively determine which weights to set to zero. Therefore, each weight w_i of the neural network is multiplied by a non-negative stochastic Bernoulli distributed gate g

[74] Schwarz et al. (1978), "Estimating the dimension of a model"

[1] Akaike (1998), "Information theory and an extension of the maximum likelihood principle"

$$w_i = \tilde{w}_i \cdot g_j \qquad\qquad q(g_j \mid \pi_j) = \mathrm{Ber}(\pi_j)\,. \tag{4.11}$$

The dropout rate of each weight is thus $1 - \pi_j$. The expectation of a weight being non-zero is π_j. Those gates collectively learn which weights are relevant. This is illustrated in figure 4.4, which shows a simple EQL architecture and indicates the Bernoulli random variables by a simple coin toss symbol. The equation learner infers the optimal parameters and the dropout rates of each weight. Decreasing dropout rates correspond to an augmented network capacity, while increasing dropout rates are associated with a

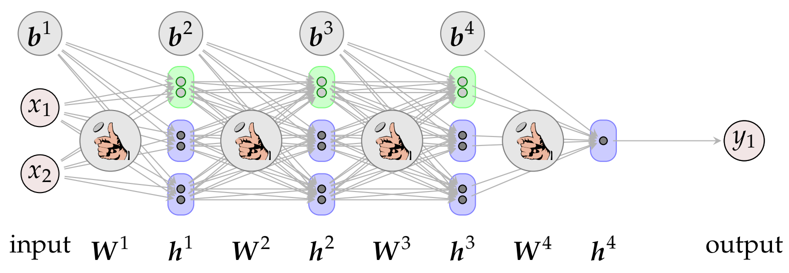


Figure 4.4: This simple EQL architecture indicates the Bernoulli gates of equation (4.12) with coin toss symbols. The equation learners infers the weights and the dropoutrates of the Bernoulli gates.

reduction in network capacity. The expected objective is then

$$\mathcal{L} = \mathbb{E}_{q(g|\pi)} \left[\frac{1}{N} \sum_{i=1}^{N} \| \boldsymbol{y}_i - \boldsymbol{f}(\boldsymbol{x}_i, \tilde{\boldsymbol{W}} \odot \boldsymbol{g}) \|_2^2 \right] + \lambda \sum_{j \leq W} \pi_j \qquad (4.12)$$

with the element-wise product $\odot$. The last term penalizes the expected count of non-zero weights. The expected data loss given by the first term is hard to minimize due to the binary Bernoulli distribution of the gates g. Appendix section A.1 outlines the smooth surrogate objective derived by Louizos et.al. [55] to apply efficient gradient based optimization. The gates are approximated with hard-sigmoid rectifications of a continuous random variable as shown in figure A.1. Therefore, they exploit the reparametrization trick [40, 69] and concrete random variables [58]. This objective is a special case of a variational bound over weights with spike and slab [65] priors and approximate posteriors[6].

[40] Kingma and Welling (2013), "Auto-Encoding Variational Bayes"
[69] Rezende, Mohamed, and Wierstra (2014), "Stochastic backpropagation and approximate inference in deep generative models"

[58] Maddison, Mnih, and Teh (2016), "The concrete distribution: A continuous relaxation of discrete random variables"
[65] Mitchell and Beauchamp (1988), "Bayesian variable selection in linear regression"

6: Further details are provided in appendix A of Louizos et.al. [55].

4.6 Conclusion

In conclusion, this section presented three approaches to improve the expressivity and training stability of equation learning neural networks. First, we introduced a robust training method that extends the hypothesis space of the network by considering atomic functions with singularities, which is a crucial step for applying these models to real-world scenarios. Secondly, we proposed the use of copy units which enable the simultaneous training of multiple architectures, significantly reducing the computational cost and encouraging simpler network designs. It is particularly relevant for the discussions in the subsequent chapter (section 5.2) on the avoidance of prohibited atomic unit combinations. Lastly, we outlined an optimization scheme to optimize L_0 regularization directly without shrinking the parameter values. This renders the final training phase, which is found in methods such as EQL and EQL$^{\div}$, unnecessary. The three contributions are part of the experimental evaluation in section 5.5. Collectively, they set a solid foundation for incorporating domain and expert knowledge into equation learning networks, which will be explored in the following chapter together with an experimental evaluation.

Equation Learning with Expert Knowledge 5

The great expressivity of the equation learner renders the hypothesis search space huge. This chapter addresses the research question *"How to utilize domain knowledge to guide the search for better equations?"* We explore strategies for integrating *expert knowledge* into both the training process and the structure of equation learning neural networks. Section 5.2 introduces methods to incorporate expert knowledge about permitted or prohibited equation components. Subsequently, section 5.3 discusses domain-dependent structured sparsity priors to improve the search for accurate yet simple equations. To adapt to realistic conditions in science and engineering, we propose the informed equation learning neural network in section 5.4. Section 5.5 demonstrates several artificial and real-world experiments from the engineering domain, in which our system learns interpretable models with high predictive power. The chapter concludes with a summary of the most important results and contributions in section 5.6.

5.1 Motivation

This section aims to provide a clear rationale for the integration of expert knowledge into the EQL architecture and describes how this approach significantly improves the selection and retrieval of symbolic expressions in various application areas.

Equation learners face challenges when dealing with an increasing variety of types of atomic units, which increases the number of hypotheses polynomially. As a result, multiple suitable equations are often found for ambiguous datasets. Our approach emphasises the integration of expert knowledge into the EQL framework to identify the true underlying equation.

Essentially, the selection of atomic unit types used by equations is guided by expert knowledge. This choice is critical and varies depending on the application domain and the characteristics of the dataset. Section 5.2 introduces constraints against certain combinations of atomic units, ensuring the system aligns with practical engineering principles and domain-specific knowledge. By allowing experts to disable certain combinations of atomic units, our system goes beyond conventional methods that only consider a fixed set of atomic unit types, and instead tailors the selection to the domain of application. This method was developed in cooperation with applied engineers to ensure relevance and applicability.

The concept of "complexity" of an equation —not the computational complexity per se, but the subjectively percieved complexity by an expert as "basic" or "involved"— depends both on the number of terms and a domain specific complexity cost of each atomic unit. To capture this aspect, typically, a weighted sum of the number of parameters is used to measure complexity. This is a commonly accepted way to introduce domain knowledge in symbolic regression, e.g. in evolutionary search [19]. Technically, this goal can be achieved through the design of a generalized sparsity regularization as we demonstrate in section 5.3. This method enables the implementation of a user-dependent weighting scheme for the complexity of atomic unit types.

[19] Dubčáková (2011), "Eureqa: software review"

In this chapter, we contribute to the advancement of symbolic regression by introducing an informed equation learning framework, denoted as iEQL. This framework allows for a large complexity of equations, which is further enhanced by the inclusion of atomic units with singularities. Additionally, it facilitates the integration of expert knowledge into the equation formulation process. We demonstrate that this enhanced approach enables the extraction of interpretable expressions from real-world datasets within the engineering domain.

5.2 Prohibited Combinations

In specific applications certain combinations of atomic units such as $\cos(\cos(\cdot))$, $\cos(\exp(\cdot))$, $\exp(\exp(\cdot))$ and $\log(\log(\cdot))$, with respective arguments can be undesirable[1], or might not make sense. *Expert knowledge* can provide such information on possible equation structures. Therefore, all forbidden connections are removed from the architecture by masking the corresponding weights, as shown in figure 5.1. That way, a domain expert decides which combinations

1: i.e. they should only be chosen rarely

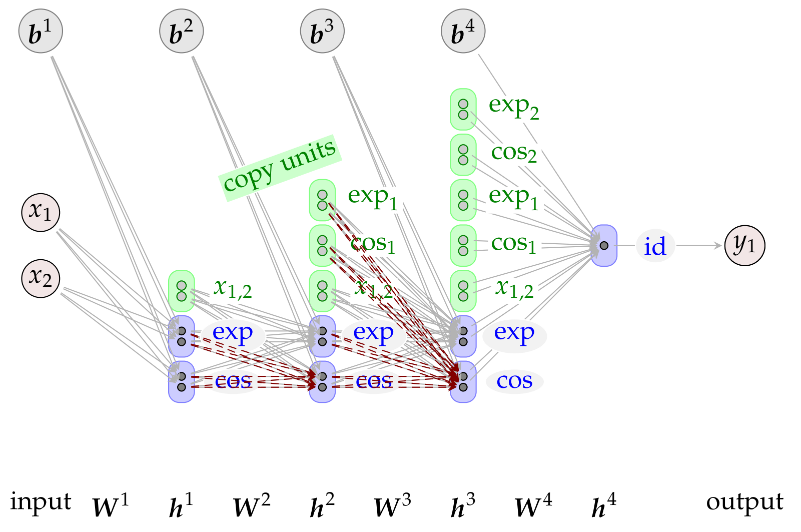


Figure 5.1: Illustration of a simple neural network architecture for learning equations with copy units. Forbidden equation components $\cos(\cos(\cdot))$ and $\cos(\exp(\cdot))$ are marked in red. The copy units contain the atomic units from all previous layers, which are also taken into account.

to use and which to exclude. Such information could not be utilized in previous frameworks with equation learning neural networks such as EQL and EQL$^{\div}$ [62, 72], which depend on identity units that accumulate outputs of previous layers in an affine transformation, which makes it impossible to omit certain combinations of functions. Our current approach, which is based on copy units, allows for the explicit exclusion of certain combinations in the iEQL.

[62] Martius and Lampert (2016), "Extrapolation and learning equations"
[72] Sahoo, Lampert, and Martius (2018), "Learning Equations for Extrapolation and Control"

5.3 Domain Specific Complexity of Atomic Units

An expert defines domain-specific complexity factors c_u for each atomic unit of type u. The complexity factors can depend on subjective criteria, like domain or personal experience, but also quite objective ones like computational cost. A specific choice of the complexity factors c_u for industrial real-world applications is shown in table 5.1. Ideally, we favor a generalized L_0 complexity measure with domain specific complexity factors

$$\mathcal{L}_\mathrm{C} = \sum_{u=1}^{U} c_u \|W_u\|_0 \,, \tag{5.1}$$

where W_u represents all weights which correspond to an atomic unit type u. The L_0 norm $\|\cdot\|_0$ counts the number of none-zero weights. Optimally, the iEQL is optimized for a Pareto optimal solution w.r.t. to data accuracy and the complexity loss $\mathcal{L}_\mathrm{C}$. This complexity measure is not differentiable and can not be used out of the box to train an EQL. Section 4.5 proposes a method to overcome this challenge through non-negative stochastic Bernoulli distributed gates based on the differentiable L_0 regularization method of Louizos et.al. [55]. Consequently, equation (4.10) can be extended to an objective loss with domain-specific complexity factors

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{W})\|_2^2 + \lambda \sum_{u=1}^{U} c_u \|W_u\|_0 \,. \tag{5.2}$$

	$*$	$/$	x^2	log	$\sqrt{\ }$	exp	cos
plain	1	1	1	1	1	1	1
motor	2	5	2	5	3	5	10

Table 5.1: Domain specific complexity factors c_u for different atomic units. No preference is denoted as *plain*. For the real-world applications (power loss of an electric machine (section 5.5.3) and torque model of a combustion engine (section 5.5.4)). We use values suggested by domain experts denoted as *motor*.

The regularization strength λ controls the relation between accuracy and complexity. Different values of λ lead to different Pareto optimal solutions. Larger values of λ trade accuracy in exchange for simplicity. Multiplying each weight by a non-negative stochastic Bernoulli distribution g with dropout rate $1 - \pi_i$ leads to the following expected objective

$$\mathcal{L} = \mathbb{E}_{q(g|\pi)} \left[\frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{f}(\boldsymbol{x}_i, \tilde{\boldsymbol{W}} \odot \boldsymbol{g})\|_2^2 \right] + \lambda \sum_{u=1}^{U} c_u \sum_{j=1}^{|W_u|} \pi_j^u \,. \tag{5.3}$$

The overall training loss is a superposition of data loss $\mathcal{L}_{\mathcal{D}}$, complexity loss $\mathcal{L}_C$ and domain penalty loss $\mathcal{L}_{\text{sf}}$ as described in equation (4.5). Necessary training phases as described in section 3.4 are an initial training without regularization, and the intermediate training phase with regularization. A final training phase to optimize the identified sparse representation without regularization is not necessary with the proposed L_0 regularization.

5.4 Informed Equation Learning Neural Network

Figure 5.2 presents the architecture of the informed equation learning neural network (iEQL) used for the experiments in section 5.5. The iEQL architecture is characterized by five hidden layers , five types of unary atomic units $\{\cos, \exp\ , \log, \sqrt{\cdot}, x^2\}$ and two types of binary atomic units $\{*, /\}$ per layer. Each atomic unit is represented four times in a layer. The final layer applies the identity operation id. The focus of this framework is on improving equation learning by incorporating expert knowledge and atomic units with singularities.

This design enables a robust representation of mathematical relationships and patterns. It includes copy units to reuse features across multiple layers and a method to incorporate expert knowledge into the training process to overcome the usual limitations of conventional equation learning neural networks. Of particular note is the model's ability to avoid forbidden combinations, thereby improving model validity and interpretability. We prohibit the following combinations:

$$\cos(\cos(\cdot)) \qquad \cos(\exp(\cdot)) \qquad \exp(\exp(\cdot)) \qquad \log(\log(\cdot)\,. \tag{5.4}$$

This is complemented by the use of Bernoulli gates for effective L_0 regularization, which helps to avoid parameter shrinkage and provides intuitive access to a domain-specific complexity measure

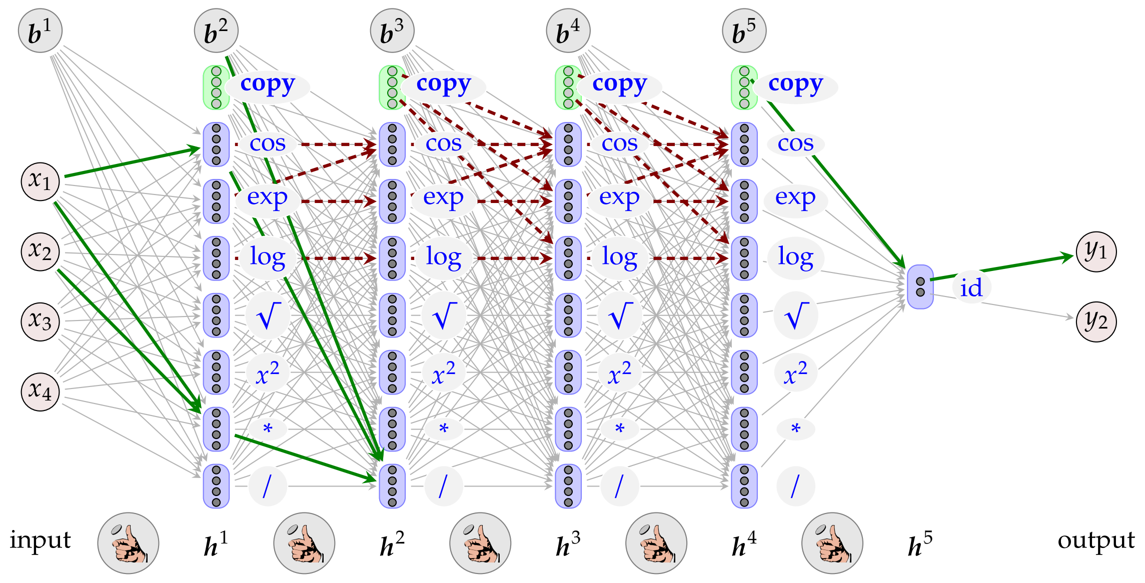


Figure 5.2: Architecture of the iEQL with five hidden layers, five types of unary atomic units {cos, exp, log, $\sqrt{}$, x^2} and two types of binary atomic units $\{*, /\}$ per layer. The copy units (see equation (4.6)) are indicated by green blocks and contain the outputs of all previous layers. The red arrows mark forbidden connections (see equation (5.4)) which are removed. The Bernoulli gates for L_0 regularization on the weights are denoted with coin toss symbols. After successful training, the iEQL represents the desired equation. The green connections visualize an exemplary model for the equation $y_2 = 0$ and $y_1 = (x_2 * x_1)/(\cos(x_1) + 1.5)$.

for experts. In particular, we differentiate between a uniform weighting scheme and a motor-specific weighting, which is used for the real-world applications in our experiments, see table 5.1.

5.5 Experiments

We demonstrate the application of the iEQL to four different use cases. The first one is to learn complex equations on simulated data of several equations. The second use case is a simulated ambiguity dataset, with which we demonstrate how to incorporate expert knowledge to influence the outcome of the equation learner. Additionally, we study the relative frequency of selected atomic units in the set of plausible equations with and without expert knowledge on a simulated dataset and a real-world dataset. Use cases three and four both address real-world applications in industry, related to determining expressions for the power loss of an electric machine and a torque model of a combustion engine. We compare the iEQL to five different algorithms:

[72] Sahoo, Lampert, and Martius (2018), "Learning Equations for Extrapolation and Control"

- EQL$^{\div}$, a state-of-the-art method from Sahoo et.al. [72] with atomic unit types $\{\sin, \cos, *, \text{identity}\}$ in each hidden layer and division in the final layer,
- a multi-layer perceptron (MLP) with tanh activation functions and five hidden layers with 50 neurons each,
- a genetic algorithm (PySR,[14]) with two different configurations GA1 and GA2,
- a Gaussian Process (GP) for the real-world datasets, calculated with ASCMO [31], which is a standard tool from the engineering domain,
- the mean predictor (MP) on the train set.

[14] Cranmer (2023), "Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl"

[31] Hoffmann, Schrott, Huber, and Kruse (2015), "Modellbasierte Methoden zur Applikation moderner Verbrennungsmotoren"

Further details on training and parameter settings are outlined in appendix section B.1. All experiments are executed five times, and we report median, minimum and maximum (in sub- and superscript) of root mean squared error on test datasets.

5.5.1 Learning Complex Equations

benchmark equations

We evaluate the performance of all five algorithms alongside the iEQL using two distinct sets of benchmark equations as outlined in table 5.2. The first set comprises equations (S0–S6), in which operations such as $\{\pm, \sin, \cos, *, /\}$ are used. The second set comprises equations (S7–S10), which also contain $\{\log, \exp\}$ operations. A two-dimensional contour plot illustrating all datasets is shown in figure 5.3. Most of these functions were compiled from various studies in the field of symbolic regression [37, 72, 79].

[37] Jin, Fu, Kang, Guo, and Guo (2019), "Bayesian symbolic regression"
[72] Sahoo, Lampert, and Martius (2018), "Learning Equations for Extrapolation and Control"
[79] Trujillo, Muñoz, Galván-López, and Silva (2016), "neat Genetic Programming: Controlling bloat naturally"

Similar to the experimental setup of Sahoo et al. [72], train datasets consist of 10^4 randomly sampled datapoints in the train domain and outputs are corrupted with standard normal noise of standard deviation $\gamma = 0.01$. For validation 10% of the train dataset is used. Test datasets $\mathcal{D}^{\text{test}}$ consist of 10^3 randomly sampled datapoints

Table 5.2: Complex equations on which the iEQL is evaluated. Most functions are gathered from different papers on symbolic regression. A star (*) indicates that input and output were scaled and shifted.

Data-set	Ground truth equation	motivated from	Domain train	Domain test
S0	$y = (1 - x_2^2)/(\sin(2\pi\, x_1) + 1.5)$		$[-1, 1]^4$	$[-2, 2]^4$
S1	$y = [\sin(\pi x_1) + \sin(2\pi x_2 + \pi/8) + x_2 - x_3 x_4]\,/3$	[72]	$[-1, 1]^4$	$[-2, 2]^4$
S2	$y = \left[\sin(\pi x_1) + x_2 \cos(2\pi x_1 + \pi/4) + x_3 - x_4^2\right]\,/3$	[72]	$[-1, 1]^4$	$[-2, 2]^4$
S3	$y = [(1 + x_2)\sin(\pi x_1) + x_2 x_3 x_4]\,/3$	[72]	$[-1, 1]^4$	$[-2, 2]^4$
S4*	$y = (3.0375\, x_1 x_2 + 5.5 \sin(9/4\,(x_1 - 2/3)(x_2 - 2/3)))/5$	[37]	$[-1, 1]^4$	$[-2, 2]^4$
S5*	$y = \frac{(5x_1)^4}{(5x_1)^4+1} + \frac{(5x_2)^4}{(5x_2)^4+1}$	[79]	$[-1, 1]^4$	$[-2, 2]^4$
S6*	$y = ((1 - x_1)^2 + (1 - x_3)^2 + 100(x_2 - x_1^2)^2$ $+100(x_4 - x_3^2)^2)/1500$	4D-Rosenbrock	$[-1, 1]^4$	$[-2, 2]^4$
S7*	$y = (1.5\mathrm{e}^{1.5x_1} + 5\cos(3\, x_2))/10$	[37]	$[-1, 1]^4$	$[-2, 2]^4$
S8*	$y = \log(2x_2 + 1) - \log(4x_1^2 + 1)$	[79]	$[0, 1]^2$	$[0, 2]^2$
S9*	$y = \frac{\exp(-(4x_1-0.7)^2)}{1.2+(4x_2-2.2)^2}$	[79]	$[0, 1]^2$	$[0, 2]^2$
S10*	$y = \frac{x_3+1}{3\pi^2} \frac{(5x_1+1)^3}{\exp\left(\frac{(x_3+1)(5x_1+1)}{(x_4+1)(0.5x_2+1)}\right)-1}$	[80]	$[0, 1]^4$	$[0, 2]^4$

from the test domain (see table 5.2). Interpolation test datasets $\mathcal{D}_{\text{int}}^{\text{test}}$ and extrapolation test datasets $\mathcal{D}_{\text{ex}}^{\text{test}}$ consist of 5000 randomly sampled datapoints from the train domain (see table 5.2). Note that for the equations S0–S6 and S7 with their four dimensional input, the extrapolation area of the test dataset is 15 times larger than its interpolation area. The included extrapolation domain is a good indicator for whether or not the ground truth has been found.

evaluation strategy

Our evaluation strategy strictly follows the strategy outlined by Sahoo et al. [72] and in particular adheres to their model selection criteria V_{int}-S and $\mathrm{V}_{\text{int\&ex}}$, the latter involving the use of 40 extrapolation points. These criteria are described in equation (3.16) and equation (3.18) respectively.

The Pareto plots for datasets (S0–S10) are depicted in figure 5.4, facilitating a comparison of V_{int}-S and $\mathrm{V}_{\text{int\&ex}}$. Each subfigure contains a classic Pareto analysis that compares the validation RMSE (v_{int}, equation (3.14)) and complexity, as well as the validation RMSE and extrapolation RMSE (v_{exp}, equation (3.17)). This dual comparison shows that the optimal extrapolation performance usually exhibits a validation RMSE at noise level, which is 0.01. The analysis highlights that the two selection criteria often identify different optimal equations.

Table 5.3: Comparative analysis of root mean squared error (RMSE) of discovered equations for (S0–S10) with selection criterion $\mathrm{V_{int}}$-S. The evaluation is divided into three different types of test datasets $\mathcal{D}^{\text{test}}$, $\mathcal{D}^{\text{test}}_{\text{int}}$, $\mathcal{D}^{\text{test}}_{\text{ex}}$. The table contains median, minimum and maximum values (in sub- and superscript) of the RMSE for each model and dataset combination. The baseline models compared include the EQL$^{\div}$ [72], a genetic algorithm (GA) [14], a multi-layer perceptron (MLP) and the mean predictor (MP) applied to the train dataset.

$\mathrm{V_{int}}$-S	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
dataset $\mathcal{D}^{\text{test}}$											
MP	1.57	0.66	0.75	0.64	1.17	0.44	0.82	0.88	1.09	0.19	0.66
MLP	$1.11^{1.17}_{1.04}$	$0.45^{0.54}_{0.35}$	$0.41^{0.42}_{0.39}$	$0.36^{0.37}_{0.34}$	$0.65^{0.70}_{0.57}$	$0.07^{0.15}_{0.02}$	$0.65^{0.66}_{0.65}$	$0.67^{0.70}_{0.64}$	$0.24^{0.25}_{0.22}$	$0.01^{0.02}_{0.01}$	$0.39^{0.40}_{0.37}$
GA1	$1.58^{2.45}_{1.21}$	$0.91^{0.91}_{0.91}$	$0.71^{449.47}_{0.33}$	$0.79^{1.24}_{0.79}$	$1.34^{1.46}_{1.18}$	$0.44^{0.44}_{0.44}$	$0.82^{0.82}_{0.82}$	$0.82^{1.11}_{0.49}$	$0.62^{1.19}_{0.31}$	$0.19^{0.48}_{0.14}$	$0.66^{0.66}_{0.59}$
GA2	$1.22^{3.57}_{0.72}$	$0.66^{1.35}_{0.57}$	$0.60^{0.72}_{0.58}$	$0.62^{1.40}_{0.59}$	$0.96^{0.98}_{0.96}$	$0.34^{0.39}_{0.12}$	$1.72^{27989.31}_{0.67}$	$0.82^{1.02}_{0.57}$	$0.24^{1.21}_{0.24}$	$1.46^{3.63}_{0.12}$	$0.42^{0.65}_{0.28}$
EQL$^{\div}$	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.09}_{0.01}$	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$	$2.15^{11.23}_{0.55}$	$0.14^{0.36}_{0.07}$	$0.03^{0.05}_{0.01}$	$0.28^{1.75}_{0.23}$
iEQL	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.39}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.03}_{0.01}$	$0.01^{0.95}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.37^{0.37}_{0.37}$	$0.88^{0.94}_{0.86}$	$0.13^{0.13}_{0.13}$	$0.24^{0.64}_{0.20}$
dataset $\mathcal{D}^{\text{test}}_{\text{int}}$											
MP	0.46	0.36	0.35	0.28	0.62	0.48	0.06	0.38	0.60	0.22	0.12
MLP	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
GA1	$1.63^{2.35}_{1.24}$	$0.91^{0.91}_{0.91}$	$0.72^{414.26}_{0.33}$	$0.80^{1.24}_{0.80}$	$1.39^{1.50}_{1.20}$	$0.44^{0.44}_{0.44}$	$0.83^{0.83}_{0.83}$	$0.81^{1.11}_{0.49}$	$0.69^{1.32}_{0.33}$	$0.17^{0.53}_{0.12}$	$0.70^{0.70}_{0.63}$
GA2	$1.25^{3.98}_{0.73}$	$0.70^{1.41}_{0.58}$	$0.62^{0.74}_{0.60}$	$0.63^{1.37}_{0.60}$	$0.99^{1.01}_{0.99}$	$0.34^{0.42}_{0.12}$	$1.79^{105589.11}_{0.80}$	$0.81^{1.01}_{0.57}$	$0.27^{1.34}_{0.27}$	$1.64^{3.96}_{0.12}$	$0.46^{0.78}_{0.30}$
EQL$^{\div}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
iEQL	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.02}_{0.02}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$
dataset $\mathcal{D}^{\text{test}}_{\text{ex}}$											
MP	1.62	0.67	0.76	0.65	1.19	0.43	0.83	0.86	1.21	0.18	0.70
MLP	$1.13^{1.19}_{1.06}$	$0.46^{0.56}_{0.36}$	$0.43^{0.44}_{0.41}$	$0.36^{0.37}_{0.34}$	$0.70^{0.75}_{0.60}$	$0.06^{0.15}_{0.02}$	$0.66^{0.67}_{0.65}$	$0.67^{0.69}_{0.64}$	$0.26^{0.27}_{0.23}$	$0.02^{0.02}_{0.01}$	$0.43^{0.44}_{0.42}$
GA1	$1.56^{2.26}_{1.19}$	$0.89^{0.89}_{0.89}$	$0.70^{27.57}_{0.37}$	$0.80^{1.34}_{0.80}$	$1.21^{1.31}_{1.05}$	$0.40^{0.40}_{0.40}$	$0.78^{0.78}_{0.78}$	$0.89^{1.18}_{0.51}$	$0.67^{1.36}_{0.31}$	$0.17^{0.52}_{0.11}$	$0.94^{0.94}_{0.85}$
GA2	$1.23^{3.13}_{0.75}$	$0.73^{1.35}_{0.55}$	$0.57^{0.72}_{0.55}$	$0.58^{1.40}_{0.57}$	$0.89^{0.91}_{0.89}$	$0.31^{0.44}_{0.08}$	$0.75^{38e3}_{0.64}$	$0.89^{1.12}_{0.59}$	$0.26^{1.38}_{0.26}$	$1.52^{4.21}_{0.13}$	$0.56^{0.75}_{0.35}$
EQL$^{\div}$	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.08}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$	$0.26^{0.26}_{0.26}$	$0.12^{4.36}_{0.08}$	$0.06^{0.15}_{0.02}$	$8.95^{72.55}_{0.36}$
iEQL	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.40}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$	$0.01^{1.00}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.36^{0.36}_{0.36}$	$0.97^{1.03}_{0.95}$	$0.15^{0.15}_{0.15}$	$0.24^{0.71}_{0.20}$

A similar analysis for EQL$^{\div}$ is given in figure 5.5. The complexity measure used in EQL$^{\div}$ counts the number of active atomic units as described by Martius and Lampert [62]. It differs from the complexity measure in iEQL as defined in equation (5.1), which is the count of all non-zero weights weighted by domain specific complexity factors.

The experimental evaluation on the test dataset $\mathcal{D}^{\text{test}}$, $\mathcal{D}^{\text{test}}_{\text{int}}$and $\mathcal{D}^{\text{test}}_{\text{ex}}$is presented in table 5.3 with the selection criterion $\mathrm{V_{int}}$-S and table 5.4 with the selection criterion $\mathrm{V_{int\&ex}}$. The interpolation dataset $\mathcal{D}^{\text{test}}_{\text{int}}$highlights the interpolation performance and the extrapolation dataset $\mathcal{D}^{\text{test}}_{\text{ex}}$emphasizes the differences in predictions of the selected equations in the extrapolation domain.
Neural network-based models (MLP, EQL$^{\div}$, iEQL) are in principle able to represent the train data perfectly with training and validation error at noise level for all datasets. This is also reflected in the RMSE of the interpolation test datasets $\mathcal{D}^{\text{test}}_{\text{int}}$. Unsurprisingly, the MLP is not able to capture the data-generating equation (ground truth) by any means on the overall test datasets $\mathcal{D}^{\text{test}}$. Therefore, it does not perform well on the test domains except for equation S9, which is mainly zero in the extrapolation domain.

Table 5.4: Comparative analysis of root mean squared error (RMSE) of discovered equations for (S0–S10) with selection criterion $\mathrm{V}_{\mathrm{int\&ex}}$. The evaluation is divided into three different types of test datasets: $\mathcal{D}^{\mathrm{test}}$, $\mathcal{D}^{\mathrm{test}}_{\mathrm{int}}$, and $\mathcal{D}^{\mathrm{test}}_{\mathrm{ex}}$. The table contains median, minimum and maximum values (in sub- and superscript) of the RMSE for each model and dataset combination. The baseline models compared include the EQL$^{\div}$ [72], a genetic algorithm (GA) [14], a multi-layer perceptron (MLP) and the mean predictor (MP) applied to the train dataset.

$\mathrm{V}_{\mathrm{int\&ex}}$	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
dataset $\mathcal{D}^{\mathrm{test}}$											
MP	1.57	0.66	0.75	0.64	1.17	0.44	0.82	0.88	1.09	0.19	0.66
MLP	$1.11^{1.17}_{1.04}$	$0.45^{0.54}_{0.35}$	$0.41^{0.42}_{0.39}$	$0.36^{0.37}_{0.34}$	$0.65^{0.70}_{0.57}$	$0.07^{0.15}_{0.02}$	$0.65^{0.66}_{0.65}$	$0.67^{0.70}_{0.64}$	$0.24^{0.25}_{0.22}$	$0.01^{0.02}_{0.01}$	$0.39^{0.40}_{0.37}$
GA1	$1.07^{1.58}_{0.91}$	$0.77^{0.77}_{0.77}$	$0.33^{0.35}_{0.01}$	$0.01^{0.47}_{0.01}$	$1.20^{1.23}_{1.07}$	$0.29^{0.29}_{0.29}$	$0.82^{0.82}_{0.82}$	$0.82^{1.11}_{0.28}$	$0.35^{0.62}_{0.26}$	$0.14^{0.19}_{0.14}$	$0.60^{0.66}_{0.56}$
GA2	$0.39^{1.47}_{0.01}$	$0.66^{0.77}_{0.01}$	$0.37^{0.77}_{0.01}$	$0.60^{1.78}_{0.58}$	$0.01^{0.01}_{0.01}$	$0.08^{0.42}_{0.01}$	$4.26^{inf}_{0.74}$	$0.28^{0.45}_{0.04}$	$0.10^{0.51}_{0.06}$	$0.10^{0.77}_{0.04}$	$0.35^{0.60}_{0.28}$
EQL$^{\div}$	$0.01^{0.12}_{0.01}$	$0.01^{0.17}_{0.01}$	$0.01^{0.05}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{1.38}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$	$0.09^{0.11}_{0.09}$	$0.09^{0.19}_{0.06}$	$0.02^{0.33}_{0.01}$	$0.36^{1.89}_{0.22}$
iEQL	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.06}_{0.01}$	$0.03^{0.05}_{0.02}$	$0.01^{0.02}_{0.01}$	$0.07^{0.13}_{0.07}$
dataset $\mathcal{D}^{\mathrm{test}}_{\mathrm{int}}$											
MP	0.46	0.36	0.35	0.28	0.62	0.48	0.06	0.38	0.60	0.22	0.12
MLP	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
GA1	$1.09^{1.63}_{0.91}$	$0.79^{0.79}_{0.79}$	$0.33^{0.36}_{0.01}$	$0.01^{0.46}_{0.01}$	$1.24^{1.29}_{1.08}$	$0.31^{0.31}_{0.31}$	$0.83^{0.83}_{0.83}$	$0.81^{1.11}_{0.27}$	$0.39^{0.69}_{0.29}$	$0.14^{0.17}_{0.12}$	$0.64^{0.70}_{0.59}$
GA2	$0.39^{1.52}_{0.01}$	$0.70^{0.79}_{0.01}$	$0.37^{0.79}_{0.01}$	$0.61^{1.75}_{0.59}$	$0.01^{0.01}_{0.01}$	$0.08^{0.42}_{0.01}$	$5.70^{inf}_{0.75}$	$0.27^{0.44}_{0.04}$	$0.11^{0.68}_{0.06}$	$0.13^{0.83}_{0.04}$	$0.35^{0.66}_{0.30}$
EQL$^{\div}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
iEQL	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
dataset $\mathcal{D}^{\mathrm{test}}_{\mathrm{ex}}$											
MP	1.62	0.67	0.76	0.65	1.19	0.43	0.83	0.86	1.21	0.18	0.70
MLP	$1.13^{1.19}_{1.06}$	$0.46^{0.56}_{0.36}$	$0.43^{0.44}_{0.41}$	$0.36^{0.37}_{0.34}$	$0.70^{0.75}_{0.60}$	$0.06^{0.15}_{0.02}$	$0.66^{0.67}_{0.65}$	$0.67^{0.69}_{0.64}$	$0.26^{0.27}_{0.23}$	$0.02^{0.02}_{0.01}$	$0.43^{0.44}_{0.42}$
GA1	$1.08^{1.56}_{0.94}$	$0.78^{0.78}_{0.78}$	$0.27^{0.38}_{0.01}$	$0.01^{0.52}_{0.01}$	$1.09^{1.14}_{0.87}$	$0.26^{0.26}_{0.26}$	$0.78^{0.78}_{0.78}$	$0.89^{1.18}_{0.35}$	$0.39^{0.67}_{0.29}$	$0.14^{0.17}_{0.11}$	$0.87^{0.94}_{0.83}$
GA2	$0.44^{1.43}_{0.01}$	$0.70^{0.78}_{0.01}$	$0.40^{0.72}_{0.01}$	$0.59^{2.29}_{0.55}$	$0.01^{0.01}_{0.01}$	$0.06^{0.38}_{0.01}$	$0.69^{\mathrm{inf}}_{0.68}$	$0.35^{0.57}_{0.06}$	$0.11^{0.45}_{0.07}$	$0.11^{0.69}_{0.04}$	$0.40^{0.66}_{0.31}$
EQL$^{\div}$	$0.01^{0.12}_{0.01}$	$0.01^{0.18}_{0.01}$	$0.01^{0.06}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{3.27}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$	$0.09^{0.10}_{0.09}$	$0.06^{0.08}_{0.03}$	$0.01^{0.02}_{0.01}$	$0.27^{0.47}_{0.17}$
iEQL	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.06}_{0.01}$	$0.04^{0.06}_{0.03}$	$0.01^{0.02}_{0.01}$	$0.08^{0.13}_{0.07}$

The extrapolation dataset $\mathcal{D}^{\mathrm{test}}_{\mathrm{ex}}$emphasizes this effect.

In contrast, GA, EQL$^{\div}$, and iEQL learn meaningful equations, which extrapolate. If the correct equation has been identified, the test RMSE is expected to be at noise level. The genetic algorithm (GA1) captures the data generating function on the datasets S2 and S3 at least in one experiment and GA2 on the datasets S0, S1, S2, S4 and S5.
On datasets S0–S6 the iEQL captures the data generating function reliably for all five runs. Despite its larger expressivity, with $\{\cos, \exp\ , \log, \sqrt{\ }, x^2, *, /\}$ atomic units, it even outperforms the EQL$^{\div}$ architecture, which uses just the necessary atomic units $\{\cos, \sin, *, /\}$. Equations S7–S10 are more difficult to learn. iEQL outperforms all baselines with $\mathrm{V}_{\mathrm{int\&ex}}$, but with the complexity based selection criterion $\mathrm{V}_{\mathrm{int}}$-S it does not consistently select the optimal equation.

Both criteria $\mathrm{V}_{\mathrm{int\&ex}}$ and $\mathrm{V}_{\mathrm{int}}$-S perform similarly on equations (S0–S6), but the $\mathrm{V}_{\mathrm{int\&ex}}$ method performs better on the more sophisticated equations (S7–S10). However, it requires additional extrapolation datapoints, which might not be available in real-world datasets. Therefore, we apply the $\mathrm{V}_{\mathrm{int}}$-S method on the two real-world datasets.

Table 5.5: Comparative analysis of root mean squared error (RMSE) of discovered equations for (S0–S4) with selection criteria V_{int}-S and $V_{\text{int\&ex}}$ on test datasets $\mathcal{D}^{\text{test}}$. The compared models are EQL$^{\div}$ [72], iEQL and iEQL$^{0.2\cos}$. The latter model has a reduced complexity factor of 0.2 for the cosine atomic units. The table contains median, minimum and maximum values (in sub and superscript) of the root mean squared error (RMSE) for each model and dataset combination.

V_{int}-S	S0	S1	S2	S3	S4
EQL$^{\div}$	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.09}_{0.01}$	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$
iEQL	$0.01^{0.02}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.39}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.02^{0.03}_{0.01}$
iEQL$^{0.2\cos}$	$0.07^{0.28}_{0.02}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$
$V_{\text{int\&ex}}$					
EQL$^{\div}$	$0.01^{0.12}_{0.01}$	$0.01^{0.17}_{0.01}$	$0.01^{0.05}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{1.38}_{0.01}$
iEQL	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
iEQL$^{0.2\cos}$	$0.07^{0.28}_{0.02}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.02}_{0.01}$

Additional Studies with Complexity Factors

This section analyzes the influence of different complexity factors on variants of the iEQL. The models compared include EQL$^{\div}$, the iEQL and customized versions iEQL$^{0.2\cos}$, iEQL$^{5\cos}$, iEQL$^{0.2\log}$ and iEQL$^{0.2\exp}$. These variants of iEQL introduce modified complexity factors c_u, which are mentioned in the subscript, to evaluate the impact of reducing or enhancing the complexity assigned to certain atomic units on the overall performance of the model. This comparative approach provides insights into how variations in parametrization affect the accuracy reflected in the RMSE.

Table 5.5 presents a comparative analysis that focuses on the performance of iEQL$^{0.2\cos}$ on the datasets (S0–S4). All those equations contain cosine functions as can be seen in table 5.2. This iEQL variant introduces a modified complexity factor of 0.2 for cosine atomic units in order to favor cosine atomic units. A more detailed study of the complexity factor strength and its relation to the relative frequency of atomic units is provided in section 5.5.2. The evaluation is based on two different selection criteria for V_{int}-S and $V_{\text{int\&ex}}$. The complexity based criterion is of higher interest, since it requires a "good" interpretation of complexity, where the extrapolation based criterion ignores complexity. The cosine preference leads to better results for equations S2 and S3 when applying the selection criterion V_{int}-S. However, for equation S0, the results are less favorable.

Equation S5, which contains no periodic functions, equation S7, which contains an exponential function, and equation S8, which contains a logarithmic function, are of particular interest. In this context, variants of iEQL (iEQL$^{5\cos}$ and iEQL$^{0.2\exp}$ for S5 and S7 respectively; iEQL$^{0.2\log}$ for S8) are evaluated against the iEQL and the EQL$^{\div}$ standard configurations in table 5.6, which lists various complexity factors together with the equations to be analyzed.

For equation S5, the penalty on periodic functions leads to a significant improvement in the maximum RMSE under the V_{int}-S

	V_{int}-S			$V_{\text{int\&ex}}$		
dataset	$\mathcal{D}^{\text{test}}_{\text{int}}$	$\mathcal{D}^{\text{test}}_{\text{ex}}$	$\mathcal{D}^{\text{test}}$	$\mathcal{D}^{\text{test}}_{\text{int}}$	$\mathcal{D}^{\text{test}}_{\text{ex}}$	$\mathcal{D}^{\text{test}}$
equation S5						
EQL$^{\div}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
iEQL	$0.01^{0.01}_{0.01}$	$0.01^{1.00}_{0.01}$	$0.01^{0.95}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
iEQL$^{5\cos}$	$0.03^{0.04}_{0.01}$	$0.02^{0.35}_{0.01}$	$0.02^{0.35}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
equation S7						
EQL$^{\div}$	$0.01^{0.01}_{0.01}$	$0.26^{0.26}_{0.26}$	$2.15^{11.23}_{0.55}$	$0.01^{0.01}_{0.01}$	$0.09^{0.10}_{0.09}$	$0.09^{0.11}_{0.09}$
iEQL	$0.02^{0.02}_{0.02}$	$0.36^{0.36}_{0.36}$	$0.37^{0.37}_{0.37}$	$0.01^{0.01}_{0.01}$	$0.02^{0.06}_{0.01}$	$0.02^{0.06}_{0.01}$
iEQL$^{0.2\exp}$	$0.02^{0.02}_{0.01}$	$0.36^{0.36}_{0.01}$	$0.37^{0.37}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.03^{0.08}_{0.01}$	$0.03^{0.08}_{0.01}$
equation S8						
EQL$^{\div}$	$0.01^{0.01}_{0.01}$	$0.12^{4.36}_{0.08}$	$0.14^{0.36}_{0.07}$	$0.01^{0.01}_{0.01}$	$0.06^{0.08}_{0.03}$	$0.09^{0.19}_{0.06}$
iEQL	$0.01^{0.01}_{0.01}$	$0.97^{1.03}_{0.95}$	$0.88^{0.94}_{0.86}$	$0.01^{0.01}_{0.01}$	$0.04^{0.06}_{0.03}$	$0.03^{0.05}_{0.02}$
iEQL$^{0.2\log}$	$0.01^{0.01}_{0.01}$	$1.03^{1.45}_{0.64}$	$0.94^{1.34}_{0.58}$	$0.01^{0.01}_{0.01}$	$0.03^{0.03}_{0.02}$	$0.03^{0.03}_{0.02}$

Table 5.6: This table presents a comparative analysis of RMSE for equations S5, S7 and S8, applying two different types of selection criteria, V_{int}-S and $V_{\text{int\&ex}}$. The table contains median, minimum and maximum RMSE values (in sub- and superscript) of different test datasets, $\mathcal{D}^{\text{test}}_{\text{int}}$, $\mathcal{D}^{\text{test}}_{\text{ex}}$ and $\mathcal{D}^{\text{test}}$, which represent the interpolation, extrapolation and overall performance, respectively. The evaluated models include the EQL$^{\div}$, iEQL and iEQL variants with adjusted complexity factors for cosine, exponential and logarithmic units (iEQL$^{5\cos}$ and iEQL$^{0.2\exp}$ for S5 and S7 respectively; iEQL$^{0.2\log}$ for S8).

selection criterion compared to uniform complexity factors. However, it can be observed that the median RMSE performance does not exceed that of the baseline models. In the case of equation S7, by favoring exponential atomic units, the corresponding iEQL$^{0.2\exp}$ model successfully identifies equations that perform well at the noise level under both the V_{int}-S and $V_{\text{int\&ex}}$ selection criterion. This performance is not achieved by the standard iEQL model under $V_{\text{int\&ex}}$.

The analysis of equation S8 shows improvement in performance using the selection criterion $V_{\text{int\&ex}}$ over V_{int}-S. None of the evaluated methods were able to identify an optimal equation. The EQL$^{\div}$ showed the best performance with the selection criterion V_{int}-S and less favorable results with the selection criterion $V_{\text{int\&ex}}$. Whereas iEQL$^{0.2\log}$ showed less favorable results with selection criterion V_{int}-S and the best performance with the selection criterion $V_{\text{int\&ex}}$. This highlights the subtle effects of domain specific complexity factors on equation learning.

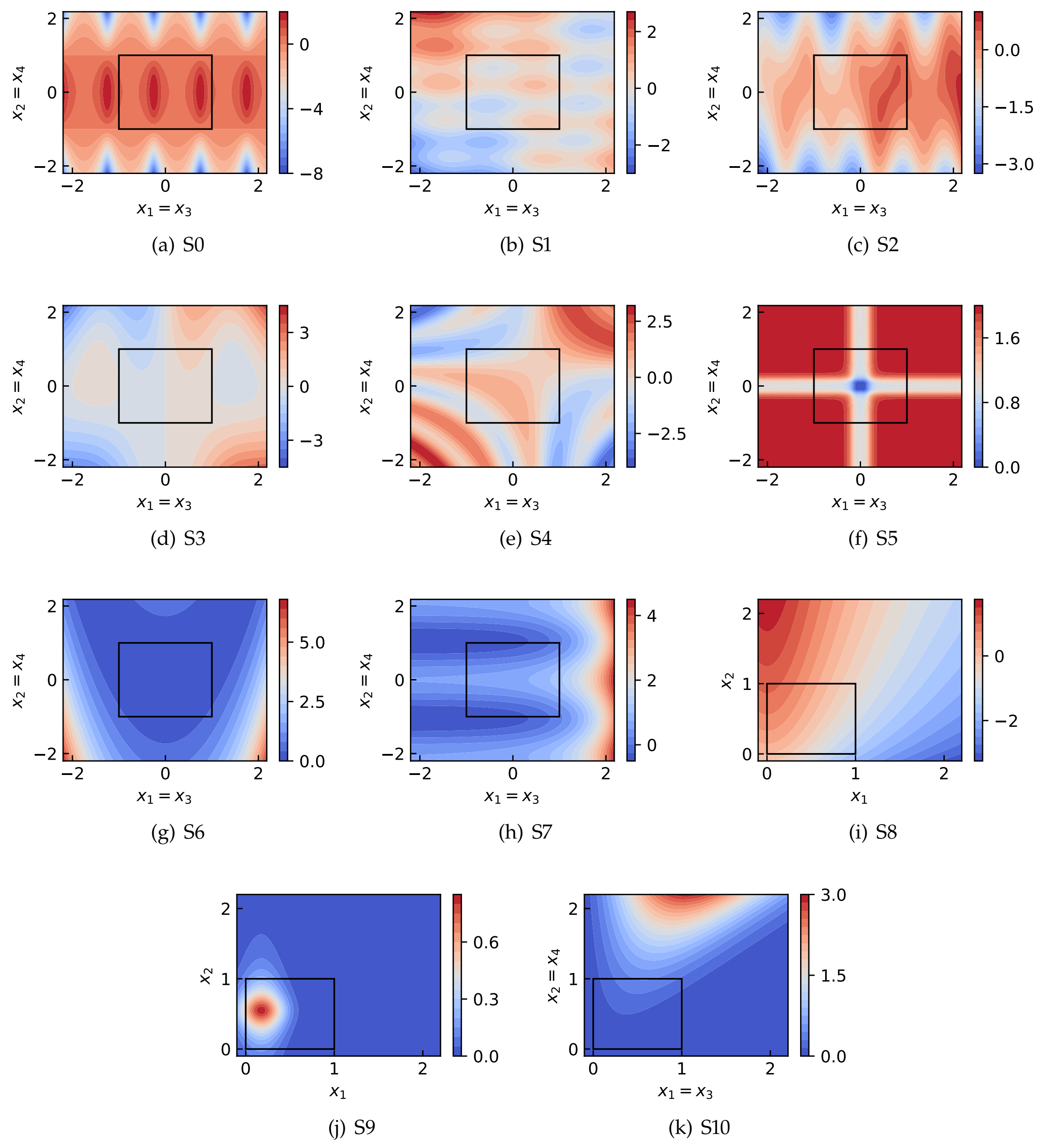


Figure 5.3: Two dimensional slices through the input space of simulated equations S0–S10 from table 5.2. The black rectangle indicates the train domain.

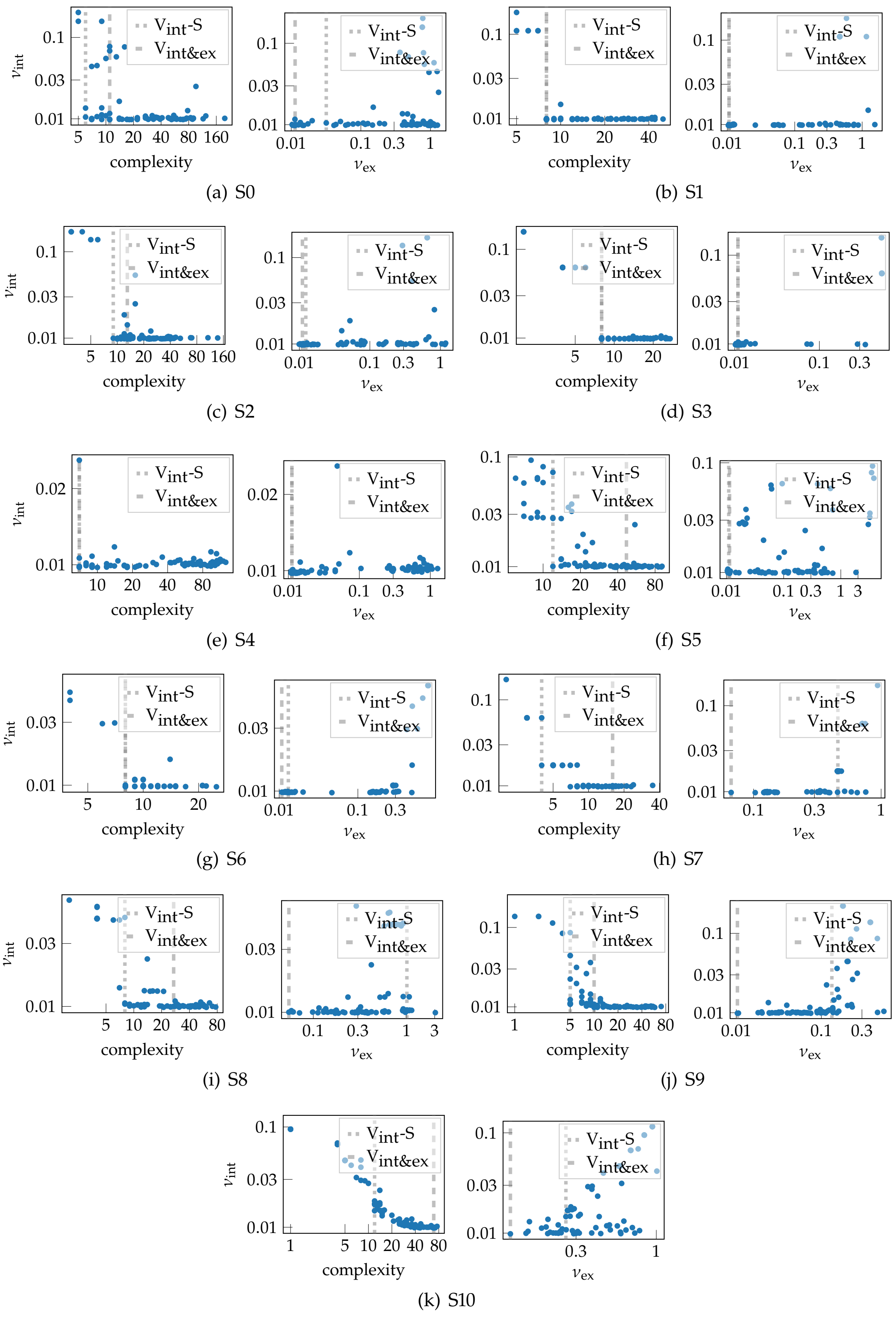


Figure 5.4: Pareto plots of iEQL representing datasets (S0–S10) for two different selection criteria: V_{int}-S and $V_{int\&ex}$. Vertical grey lines indicate the selected equations by each criterion, distinguished by dotted lines for V_{int}-S and dashed lines for $V_{int\&ex}$. Each subfigure contains two figures. The left figure compares validation RMSE against complexity, while the right figure illustrates the relationship between validation RMSE and extrapolation RMSE.

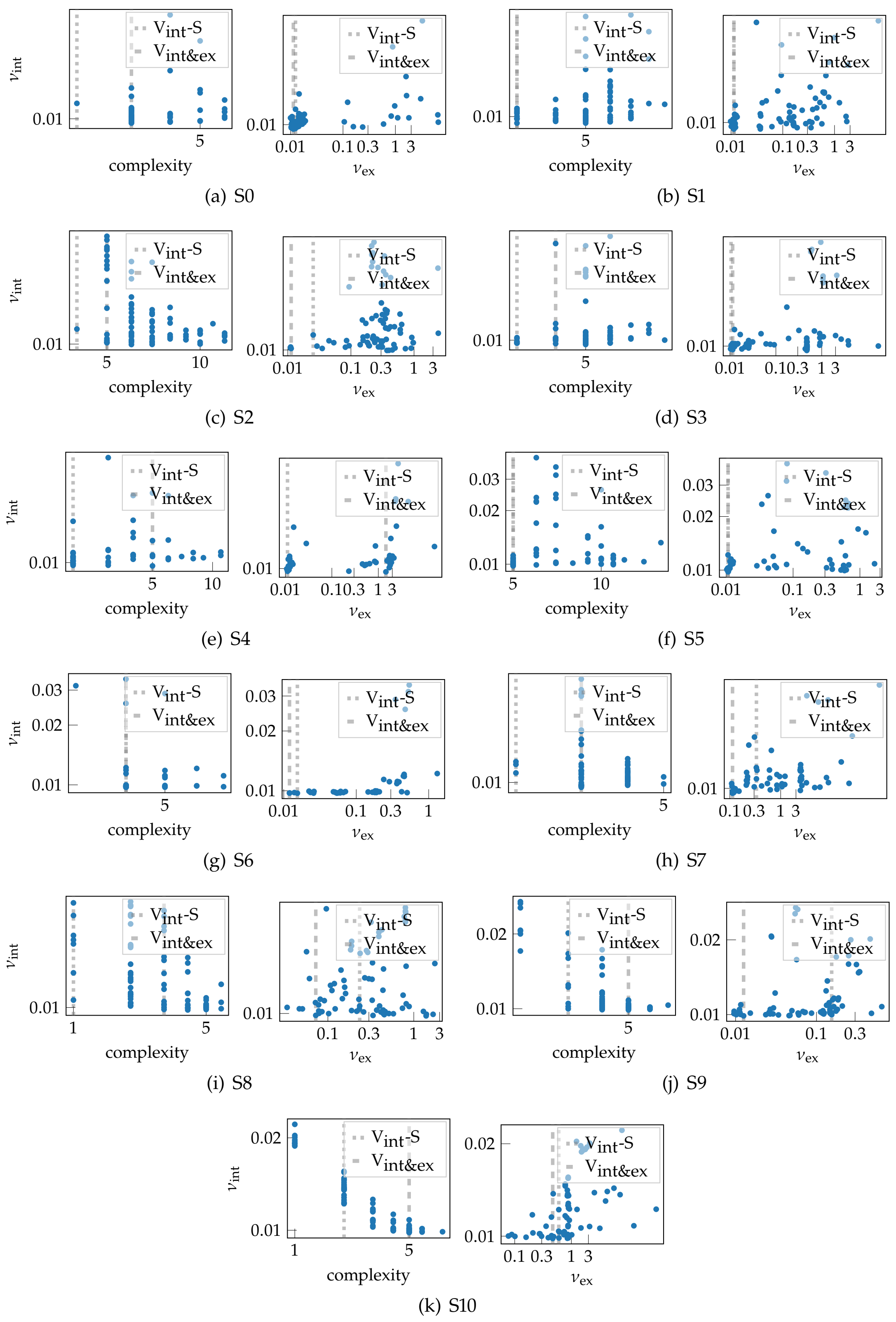


Figure 5.5: Pareto plots of EQL$^{\div}$ representing datasets (S0–S10) for two different selection criteria: V_{int}-S and $V_{int\&ex}$. Vertical grey lines indicate the selected equations by each criterion, distinguished by dotted lines for V_{int}-S and dashed lines for $V_{int\&ex}$, respectively. Each subfigure contains two figures. The left figure compares validation RMSE against complexity, while the right figure illustrates the relationship between validation RMSE and extrapolation RMSE. The complexity measure used in EQL$^{\div}$ counts the number of active atomic units as described by Martius and Lampert [62]. It differs from the complexity measure in iEQL as defined in equation (5.1).

5.5.2 Expert Knowledge for Ambiguous Data

Expert knowledge is especially important if there are several possible equations for an ambiguous dataset. Incorporating this knowledge into the complexity measure as well as the regularization can decide whether the right equation is found. The following experiment studies how different sets of complexity factors affect the relative frequency of atomic units in the set of plausible equations as well as the importance of complexity factors for the selection criterion $\mathrm{V_{int}}$-S, which is applied to the set of plausible equations. The selection criterion $\mathrm{V_{int\&ex}}$ is not suited for this experiment, since it ignores the model complexity. In order to construct an ambiguous dataset, we choose the ground truth

$$y = 8\cos(0.5\,x) - 7.5 + \epsilon\,, \quad \epsilon \sim \mathcal{N}(0, 0.01^2) \tag{5.5}$$

which, close to $x = 0$, can be modeled equivalently by a cos function or a polynomial of degree 2 as shown in figure 5.6). The output is corrupted with Gaussian noise $\mathcal{N}(0, 0.01^2)$. The train dataset consists of 10^4 randomly sampled datapoints.

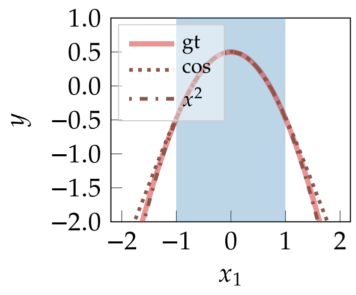


Figure 5.6: This figure illustrates the ground truth function (thick line) for the ambiguous dataset as described in equation (5.5). Also depicted are two potential solutions: a cosine function (dotted line) and a second-degree polynomial equation (dash-dotted line). The blue shading indicates the train domain.

We study nine configurations that prefer a periodic structure $c_{\cos} \in \{1/2, 1/3, \ldots, 1/10\}$ and nine configurations that favor a polynomial structure $c_{x^2} \in \{1/2, 1/3, \ldots, 1/10\}$ as well as the uniform configuration. For each configuration, 78 equations with varying levels of regularization strength are calculated, as stated in the train details in appendix section B.1. The relative frequency of cos and x^2 units for each equation is calculated for each complexity cost ratio $r = c_{\cos}/c_{x^2}$. Figure 5.7 displays the corresponding average relative frequencies. The relative frequencies follow the prior. Ratios with $r < 1$ prefer cos units and ratios with $r > 1$ prefer x^2 units. A polynomial solution seems very plausible without

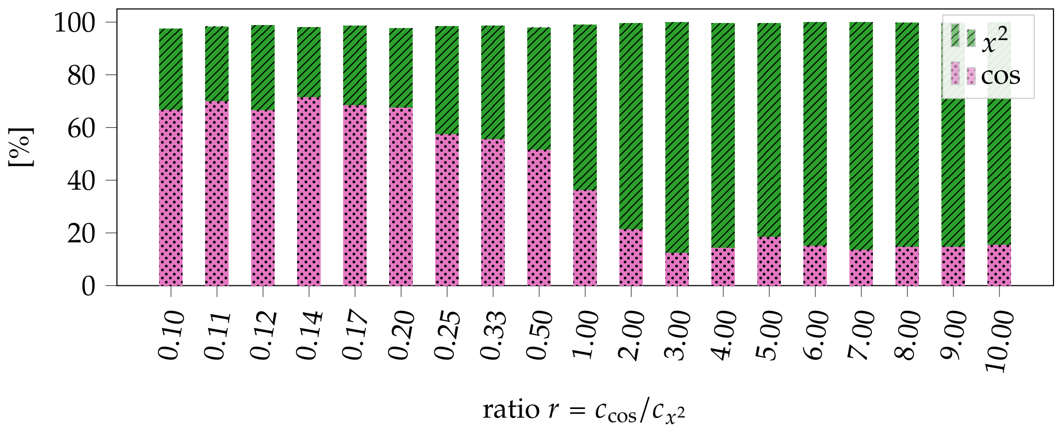


Figure 5.7: This figure illustrates the impact of expert knowledge on an ambiguous dataset, with the ground truth defined by equation (5.5). It displays the average relative frequencies of the atomic units cos and x^2 for various complexity cost ratios $r = c_{\cos}/c_{x^2}$. These frequencies are calculated for each found equation.

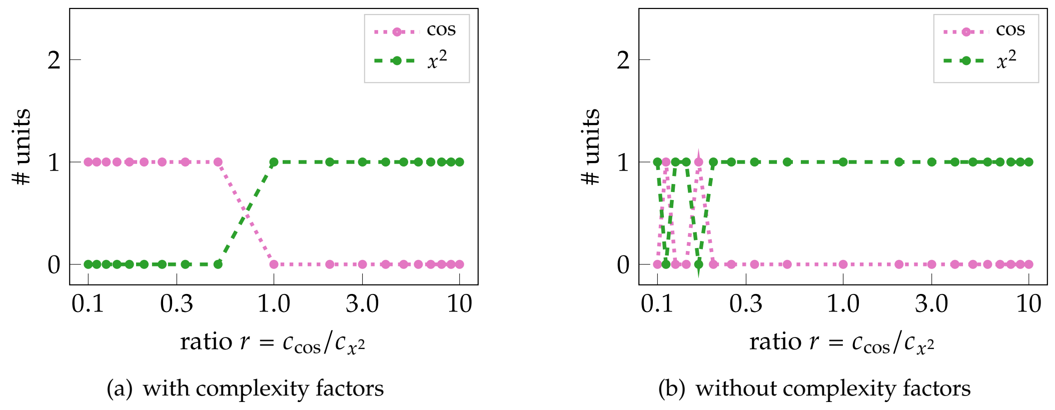


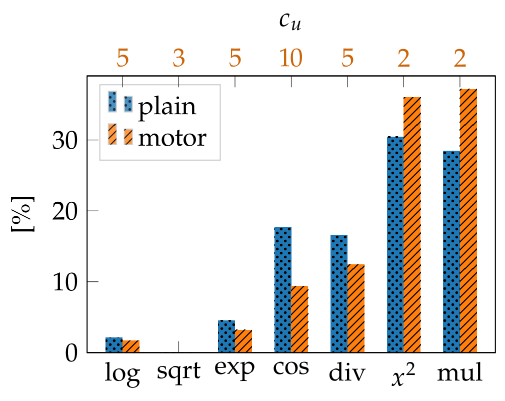
Figure 5.8: This figure illustrates the impact of the selection criterion V_{int}-S with expert knowledge on an ambiguous dataset. The left panel shows the number of cos and x^2 units that appear in the selected equation of each complexity cost ratio. Here, the left panel is selected with V_{int}-S with complexity factors and the right panel illustrates the same quantity without complexity factors in V_{int}-S.

any further information ($r = 1$). It turns out that iEQL does not converge to all kinds of simple ambiguous solutions with equal probability. Certain solutions are more likely to converge than others. In this experiment, for instance, iEQL predominantly uses x^2 units without a biased preference ($r = 1$).

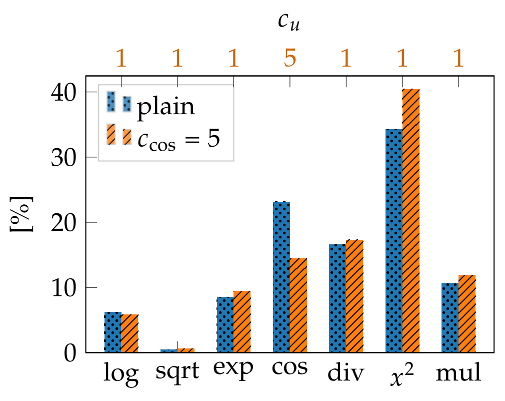
In the next part, the selection criterion is applied to the 78 found equations for each complexity cost ratio. Figure 5.8 shows the absolute frequency of cos and x^2 units that appear in the selected equation. For ratios with $r < 1$ a periodic equation is preferred and for ratios with $r > 1$ a polynomial equation is preferred. As desired, the prior is followed in the case of such a strong ambiguity. Without complexity factors, a polynomial solution is usually selected as shown in figure 5.8(a). Therefore, complexity factors are crucial to select the correct equation with respect to the prior. Since in this case both types of plausible equations have the same complexity, we found that the Adam optimizer converges better for x^2 atomic units. This leads to a preference for polynomial solutions.

Figure 5.9(a) shows the impact of motor specific complexity factors (see table 5.1) on the relative frequencies of atomic units evaluated on the set of plausible equations[2] on the real-world dataset *'torque of an internal combustion engine'*.The motor specific complexity factors clearly reduce the relative frequency of $\{\exp, \cos, /\}$ atomic units and enhances polynomial structures with $\{x^2, *\}$ units. This coincides with the expected behavior. The unit $\sqrt{\ }$ was hardly chosen, which indicates that this atomic function does not seem to be relevant for the description of the dataset. Figure 5.9(b) shows the impact of penalizing cos units with ($c_{\cos} = 5$) on a more sophisticated synthetic dataset (S5). The relative frequency of the cosine units is reduced and the relative frequency of the x^2 units is enhanced.

2: drawing from five experiments, each with 78 different regularization strength, see appendix section B.1)



(a) combustion engine



(b) S5 dataset

Figure 5.9: Relative frequency of atomic units with and without domain specific factors over the set of plausible equations (five experiemnts, each with 78 different regularization strength, see appendix section B.1) of the combustion engine experiments in (a) and of the S5 dataset experiments in (b). Domain specific complexity factors c_u are shown in the upper x-axis.

5.5.3 Power Loss of an Electric Machine

In this section, we address with the power loss of an electric machine – a relevant problem in the industrial domain [8]. Increasing requirements on control of electric machines in automotive power trains lead to the necessity of more and more sophisticated and complex models to describe the system. Those models have to be able to capture non-linear effects for different components in the system, as in the dataset for the power loss of an electric machine. Further, those models have to be minimal in computational effort and memory demand due to embedded hardware and latency constraints in the range of micro-seconds.

[8] Buchner, Boblest, Engel, Junginger, and Ulmer (2020), "An Artificial-Intelligence-Based Method to Automatically Create Interpretable Models from Data Targeting Embedded Control Applications"

In this industrial dataset, the power loss of a winding head is measured depending on the direct current I_{D}, quadratic current I_{Q}, rotor temperature T_{rot}, and motor speed n_{mtr}. The data was measured at stationary operation points. It contains 24684 datapoints recorded at equidistant variations of the quantities listed in table 5.7 within their range of operation. The test dataset contains unseen data from the entire domain and train dataset contains data only from 80% of its range of operation. Further details on data preparation are given in appendix 1.

Quantity	Test Domain	Train Domain	Description	Type
I_{D}[A]	$[1, 525]$	$[76, 451]$	direct current	input
I_{Q}[A]	$[1, 525]$	$[76, 451]$	quadratic current	input
T_{rot}[°C]	$[-20, 150]$	$[-20, 150]$	rotor temperature	input
n_{mtr}[rpm]	$[1, 16000]$	$[2001, 14001]$	motor speed	input
P_{mod}[W]	$[0, 4558]$	$[95, 3364]$	power loss	output

Table 5.7: Data properties for the power loss of an electric machine. The train domain covers just 80% of the range of operation, except for rotor temperature T_{rot}, which has only three different operation points.

Table 5.8: Results on real-world datasets. Reported here are median, minimum and maximum (in sub and superscript) of root mean squared error (RMSE) on the real-world test datasets. Domain knowledge is used for iEQL motor as given in table 5.1. We use the model selection criterion $\mathrm{V_{int}}$-S and state the number of active parameters. For the combustion engine dataset we also present the best validation models selected with $\mathrm{V_{int}}$.

	electric machine [W]		combustion engine [Nm]			
	$\mathrm{V_{int}}$-S	#param.	$\mathrm{V_{int}}$-S	#param.	$\mathrm{V_{int}}$	#param.
MP	1042.70		60.17		60.17	
GP	0.92		1.79		1.79	
GA1	$188.40\,^{347.93}_{155.84}$	$9\,^{9}_{3}$	$16.79\,^{22.37}_{13.35}$	$7\,^{11}_{5}$	$5.91\,^{7.66}_{5.00}$	$33\,^{40}_{23}$
GA2	$230.23\,^{230.23}_{230.23}$	$8\,^{8}_{8}$	$22.37\,^{22.37}_{16.79}$	$5\,^{7}_{5}$	$5.33\,^{6.29}_{4.86}$	$35\,^{39}_{34}$
EQL÷	$0.03\,^{0.04}_{0.03}$	$10\,^{14}_{7}$	$1.75\,^{1.96}_{1.55}$	$95\,^{129}_{70}$	$1.46\,^{1.61}_{1.35}$	$382\,^{427}_{225}$
iEQL	$0.03\,^{0.03}_{0.02}$	$6\,^{6}_{6}$	$2.48\,^{3.04}_{2.18}$	$65\,^{79}_{42}$	$1.44\,^{1.71}_{1.40}$	$448\,^{470}_{395}$
$\mathrm{iEQL_{motor}}$	$0.02\,^{0.02}_{0.02}$	$6\,^{6}_{6}$	$3.17\,^{6.16}_{2.90}$	$32\,^{48}_{15}$	$1.60\,^{1.79}_{1.39}$	$339\,^{386}_{170}$

We use the model selection criterion $\mathrm{V_{int}}$-S on the set of plausible equations of iEQL, EQL÷ and GA. Results are shown in table 5.8. iEQL and EQL÷ outperform the genetic algorithm (GA) and the Gaussian Process (GP) even on the train data, and provide very accurate predictions on the test set. The GP does not capture the underlying relationship, as can be seen at the large test RMSE. A closer look at the results of the genetic algorithm revealed that the internal selection criterion of GA led to competitive results: GA1 with $0.47\,^{159.11}_{0.02}$W and $19\,^{23}_{9}$parameters and GA2 with $0.02\,^{0.16}_{0.02}$W and $17\,^{19}_{15}$parameters. iEQL needs even fewer parameters than EQL÷. This is due to the use of copy units, which allow the reuse of features from previous layers directly. We emphasize that, out of the large set of possible functions, the iEQL reliably extracted a simple quadratic equation that suitably describes the dataset. To highlight the simplicity of the iEQL's result, we state in the following the structure of the selected equation, which is the same for all 5 experiments:

$$y = w_1 I_Q + w_2 I_D + w_5 (w_3 I_D + b_1)^2 + w_6 (w_4 I_Q + b_2)^2 + b_3 \,. \quad (5.6)$$

Weights are indicated with w and bias with b. All selected equations can be simplified to the same equation

$$y = 0.4 I_D^2 - 1.12 I_D + 0.41 I_Q^2 + 1.13 I_Q - 0.31 \,.$$

Numbers were rounded to three figures and input and output dimensions were anonymized. In addition, we emphasize that the algorithm learned that only two of the input variables, namely $[I_D, I_Q]$, are relevant to the output value, while the others have negligible influence.

Quantity	Unit	Min.	Max.	Description	Type
ϕ_{ex}	°Crank	−20	20	exhaust camshaft	input
ϕ_{in}	°Crank	−4	36	intake camshaft	input
r_{l}	%	13	86	relative load	input
ϕ_{ign}	°Crank	−27	61	ignition angle	input
n_{eng}	rpm	597	6000	engine speed	input
M_{eng}	Nm	−38	261	engine torque	output

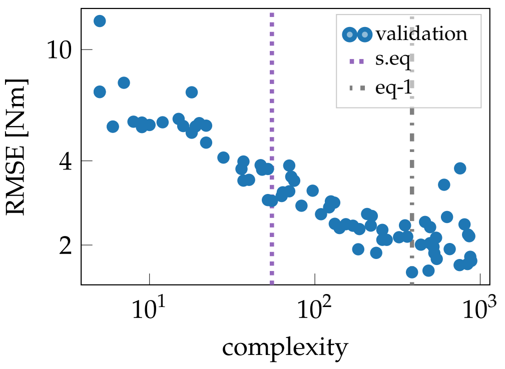
Table 5.9: Data properties for the torque model of an internal combustion engine.

5.5.4 Torque Model of an Internal Combustion Engine

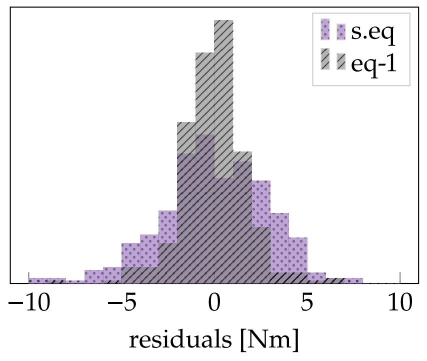
A second task from the industrial domain is the modeling of the torque of a combustion engine in dependence of the control parameters. Ongoing improvements in power train technologies require increasingly precise models for control strategies. This is commonly correlated with a complex structure of the models as well as the large number of calibration parameters. For utilization of those models in embedded systems it is essential to have fast development cycles, i.e. an automated adaptation to a new system, and low production costs. The unit costs grow with the computational needs on an embedded controller, thus computational efficiency at evaluation time is important. In practice, this often leads to a trade-off between accuracy of the model and its complexity.

In this dataset the engine torque is measured depending on exhaust camshaft ϕ_{ex}, intake camshaft ϕ_{in}, relative load r_{l}, ignition angle ϕ_{ign} and engine speed n_{eng}. The data was measured at stationary operation points. It contains 1775 datapoints recorded at variations of the quantities listed in table 5.9 within their range of operation. The dataset is split into 80% training and 20% testing data, and 10% of the train dataset is used for validation.

Instance selection can be done with the V_{int}-S criterion or solely based on validation performance as long as the equation is computationally efficient at evaluation time. We present both solutions in table 5.8 alongside with the number of active parameters. The GP is supposed to perform best on this dataset, but the EQL-networks, selected with V_{int}, perform similar or even better. With about 170-470 parameters they are fast to compute and small enough to fit on embedded systems. The EQL$^{\div}$ finds the best test RMSE results. This might indicate that for this dataset it is sufficient to deal with $\{\sin, \cos, *, \text{identity}\}$ units. Despite its larger expressive power, iEQL can compete with the EQL$^{\div}$ architecture. GA is not able to learn equations that compete on the performance level, but it can identify simple equations with about 33 parameters. Yet, $\text{iEQL}_{\text{motor}}$ also finds equations of similar complexity, but with a twice as good median test RMSE. Table 5.8 shows a noticeable relation between the performance of the models and their complexity



(a) Pareto plot



(b) histogram of residuals

Figure 5.10: Equation learning with iEQL-motor on a real-world dataset for torque of a combustion engine. Here, we show the run with median performance with the V_{int}-S criterion. Panel (a) shows the Pareto plot of the iEQL and (b) shows the histogram of residuals of the equation (s.eq) selected with the V_{int}-S criterion and a more complex equation (eq-1).

for iEQL, $\text{iEQL}_{\text{motor}}$ and $\text{EQL}^{\div}$. The more complex the model, the better its test RMSE. This effect occurs also in figure 5.10(a), which shows the Pareto plot for the $\text{iEQL}_{\text{motor}}$ experiment (median performance and V_{int}-S criterion). The histograms of residuals of the selected equation alongside the best performing equation show no significant distribution shifts, see figure 5.10(b).

We analyze the equations of median performance and the equations with the smallest number of parameters selected with the V_{int}-S criterion for iEQL and $\text{iEQL}_{\text{motor}}$ in appendix B.1.1. Their test RMSE is not as good as a GP, but their structure is interpretable. The motor specific complexity factors lead to equations of mainly polynomial structure with higher degree than without motor specific complexity factors. The latter combine polynomials with nonlinear units like $\cos, \exp, \log$ or division units. Using more nonlinear units reduces the degree of the used polynomials. This is in agreement with the analysis of relative frequencies of atomic units shown in figure 5.9(a), which clearly shows that the use of motor specific complexity factors leads to a reduction of the use of $\{\log, \cos, \exp\}$ and division units and an increase of x^2 and multiplication units.

5.6 Conclusion

We have introduced the informed equation learner iEQL. It aims to learn compact and interpretable models in form of concise mathematical equations from data. Our method differs from previous work in two key aspects: First, it is able to handle functional units with singularities in all hidden layers. Second, it can incorporate expert knowledge about the underlying system. To do so, prior knowledge is encoded in a complexity cost for individual atomic units. We evaluated our method by finding compact functional expressions for four different use cases: The iEQL performs well on established reference datasets. Customized priors can be applied to prefer certain functional terms in an ambiguous dataset supporting several possible explanations. Finally, we evaluated the algorithm on two real industrial problems. In both settings, the underlying relations of the system were unknown a priori, and the iEQL was able to extract simple, interpretable models describing the respective output variable. Depending on the needs, it also provides a high-performance solution with low computational cost.

Structured Uncertainty in Equation Learning 6

Equation learners typically do not capture uncertainty about the model or its predictions, although uncertainty is often highly structured and particularly relevant for applications in engineering and the natural sciences. This chapter addresses the research question *"How to quantify uncertainty of a set of equations that was discovered by an equation learning neural network?"*. It explores the application of simple, but effective Bayesian deep learning techniques to build structured and interpretable uncertainty over a set of plausible equations. Section 6.2 describes recent advances in the field of uncertainty quantification and equation learning and provides an overview of current methods and their limitations. Section 6.3 presents a novel approach to capturing structured uncertainty in equation learning from a Bayesian perspective, including a discussion of the relevant theoretical background, such as Gaussian regression and Laplace approximation. In section 6.4, we demonstrate applications of our method to two artificial, ambiguous datasets as well as two real-world datasets. Key findings and important contributions are summarized in section 6.5.

6.1 Motivation

In active learning, safe reinforcement learning and extrapolation tasks as well as safety critical systems like health care or automated driving, it is crucial to know about the uncertainty of the model, but, equation learners do not capture uncertainty about the model output. An equation learner discovers a set of plausible equations with different structure and varying complexity. Those equations correspond to different minima in the loss landscape. Their differences in structure can lead to a rich variety of predictions. Motivated by Occam's Razor [56] to prefer simple solutions, conventional equation learners select one equation, ignoring all others. However, if the dataset is ambiguous then there exist several similarly plausible candidate equations. Selecting one equation out of the others by chance means a loss of information about the data. This leads to overconfidence in favor of the selected equation. State-of-the-art in uncertainty quantification in deep learning are ensemble methods [22, 35, 46, 59, 66], especially to capture *global uncertainty* about the models predictions. Thus, the question arises whether we can use the set of plausible equations retrieved by an equation learner to construct an ensemble of equations for quantifying uncertainty in equation learning. Due to their differences

[56] MacKay (1992), "A practical Bayesian framework for backpropagation networks"

[22] Eschenhagen, Daxberger, Hennig, and Kristiadi (2021), "Mixtures of Laplace Approximations for Improved Post-Hoc Uncertainty in Deep Learning"
[35] Izmailov, Podoprikhin, Garipov, Vetrov, and Wilson (2018), "Averaging weights leads to wider optima and better generalization"
[46] Lakshminarayanan, Pritzel, and Blundell (2016), "Simple and scalable predictive uncertainty estimation using deep ensembles"
[59] Maddox, Izmailov, Garipov, Vetrov, and Wilson (2019), "A simple baseline for bayesian uncertainty in deep learning"
[66] Pearce, Leibfried, and Brintrup (2020), "Uncertainty in neural networks: Approximately bayesian ensembling"

in structure, the predictions of the equations strongly differ in regions that are not sufficiently covered by measurements. This is of special interest in extrapolation tasks, where no train data is available. Indeed, incorporating equations of different complexity and structure leads to sophisticated, structured uncertainty estimates. We apply a Laplace approximation to each equation in order to capture *local uncertainty* about the parameters within each equation structure. Recent successes have been shown in quantifying uncertainty in deep learning [17, 24, 34, 42, 43, 70].

[17] Daxberger, Nalisnick, Allingham, Antoran, and Hernandez-Lobato (2021), "Bayesian Deep Learning via Subnetwork Inference"
[24] Foong, Li, Hernández-Lobato, and Turner (2019), "'In-Between'Uncertainty in Bayesian Neural Networks"
[34] Immer, Korzepa, and Bauer (2021), "Improving predictions of Bayesian neural nets via local linearization"
[42] Kristiadi, Hein, and Hennig (2020), "Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks"
[43] Kristiadi, Hein, and Hennig (2020), "Learnable Uncertainty under Laplace Approximations"
[70] Ritter, Botev, and Barber (2018), "A scalable Laplace approximation for neural networks"

This work shows the application of uncertainty quantification on a set of equations that was discovered by deep equation learning. In principle, it can be applied to any set of equations found by an equation learning algorithm. We demonstrate that simple but effective forms of Bayesian deep learning can be used to build such structured and interpretable uncertainty over a set of plausible equations. In particular, we use a mixture of Laplace approximations, where each mixture component captures a different equation structure, and the local Laplace approximations capture parametric uncertainty within one family of equations.

6.2 Related work

The following section addresses the intersection of uncertainty quantification and equation learning, highlighting recent advances.

Section 4.2 gives a comprehensive overview of existing methods related to equation learning techniques. These approaches generate various plausible equations covering a range of complexities from which an optimal solution is typically selected based on predefined criteria or at the user's discretion. However, a common limitation of these methods is that they do not incorporate structured uncertainty across the set of potential equations.

Automatic statistician: Structured and interpretable uncertainty has been studied with Gaussian processes in the context of kernel learning [20, 52, 77, 88]. Especially, the *automatic statistician* aims to learn an interpretable structure of base kernels to describe high-level properties like smoothness, trends, periodicity and change points. Learning such structural forms of the kernel also enables for long-range extrapolation. We achieve similar statistical descriptions, yet with interpretable equations instead of non-parametric, black box Gaussian processes.

[20] Duvenaud, Lloyd, Grosse, Tenenbaum, and Zoubin (2013), "Structure Discovery in Nonparametric Regression through Compositional Kernel Search"
[52] Lloyd, Duvenaud, Grosse, Tenenbaum, and Ghahramani (2014), "Automatic Construction and Natural-Language Description of Nonparametric Regression Models"
[77] Sun, Zhang, Wang, Zeng, Li, and Grosse (2018), "Differentiable compositional kernel learning for Gaussian processes"
[88] Wilson, Gilboa, Nehorai, and Cunningham (2014), "Fast Kernel Learning for Multidimensional Pattern Extrapolation"

Uncertainty estimation for equations: Recent insights in uncertainty estimation with Laplace approximations for neural networks can also be applied to uncertainty estimation for equations, without the disadvantage of huge parameter spaces. The Laplace approximation for neural networks has been first introduced by MacKay [56]. It requires to invert the full Hessian, which is huge for modern neural networks. With recent developments in Hessian approximation [5, 60, 61] the Laplace approximation became accessible to modern neural networks [17, 43, 70]. Foong et al. [24] empirically show that the linearized Laplace approximation leads to better calibrated uncertainty estimates of simple neural networks than without linearization. We experienced similar effects with equations instead of neural networks. Immer et al. [34] propose to apply the Laplace approximation to the local linearization of the neural network justifying the GGN approximation. But, the Laplace approximation is prone to underestimate the uncertainty.

[5] Botev, Ritter, and Barber (2017), "Practical Gauss-Newton optimisation for deep learning"
[60] Martens (2020), "New Insights and Perspectives on the Natural Gradient Method"
[61] Martens and Grosse (2015), "Optimizing Neural Networks with Kronecker-factored Approximate Curvature"

Lakshminarayanan et al. [46] present a simple, yet state-of-the-art, method for uncertainty estimation in deep learning combining several independently trained neural networks in a deep ensemble. Motivated by their results we propose to use a mixture of Laplace approximations (MoLA,[22]) for a set of plausible equations.

[46] Lakshminarayanan, Pritzel, and Blundell (2016), "Simple and scalable predictive uncertainty estimation using deep ensembles"

[22] Eschenhagen, Daxberger, Hennig, and Kristiadi (2021), "Mixtures of Laplace Approximations for Improved Post-Hoc Uncertainty in Deep Learning"

6.3 A Bayesian Perspective on Equation Learning

An equation learner finds several plausible equations of varying structure and complexity. They correspond to different modes of the likelihood and lead to a variety in predictions, particularly in data-sparse extrapolation regions. We identify this aspect as *global uncertainty* that we capture with a mixture of local Laplace approximations (MoLA [22]). We capture the parametric uncertainty within one equation with a local Laplace approximation and refer to it as *local uncertainty*. Therefore, we consider equation learning with a *Gaussian regression* setting with a parameterized analytical equation $f_{\boldsymbol{\theta}}$, to map a d-dimensional input $\boldsymbol{x}$ to a d'-dimensional output $\boldsymbol{y}$ given by

[22] Eschenhagen, Daxberger, Hennig, and Kristiadi (2021), "Mixtures of Laplace Approximations for Improved Post-Hoc Uncertainty in Deep Learning"

Gaussian regression

$$\boldsymbol{y}_i = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + \epsilon \,, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2) \,. \tag{6.1}$$

The dataset $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}$ is assumed to be sampled iid., and it contains N datapoints each of which has Gaussian noise with variance σ^2. Assuming a zero-mean isotropic Gaussian prior $p(\boldsymbol{\theta})$ on the parameters $\boldsymbol{\theta}$ leads to the *posterior distribution*

posterior distribution

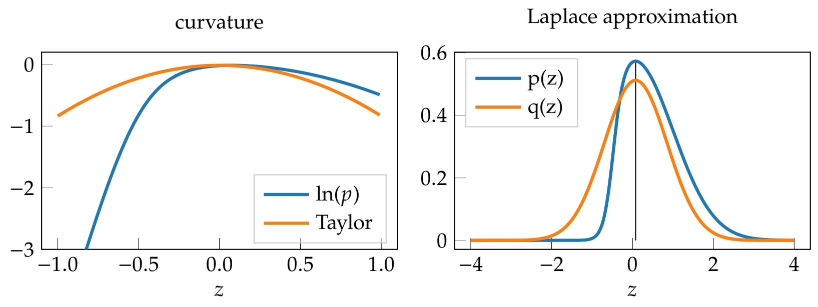


Figure 6.1: Illustration of the Laplace approximation $q(z)$ for a distribution $p(z)$ and its log-curvature of $\ln(p)$.

$$
\begin{aligned}
p(\boldsymbol{\theta} \mid \mathcal{D}) &\propto p(\mathcal{D} \mid \boldsymbol{f}_{\boldsymbol{\theta}}) p(\boldsymbol{\theta}) \\
&\propto \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \sigma^2) \mathcal{N}(\boldsymbol{\theta} \mid 0, \lambda^{-1} I)\,. \qquad (6.2)
\end{aligned}
$$

The prior $\mathcal{N}(\boldsymbol{\theta} \mid 0, \lambda^{-1})$ is related to a L_2 regularization on the parameters with a scalar precision hyperparameter λ that modulates the regularization strength. As discussed in section 2.3, the *regularized empirical risk* can be related to the *maximum a posteriori* (MAP) estimation and is thus determined by the negative log-posterior

regularized empirical risk

$$
\begin{aligned}
\mathcal{L}(\mathcal{D}, \boldsymbol{f}_{\boldsymbol{\theta}}) = &\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 + \frac{M}{2} \log(2\pi\lambda^{-1}) \\
&+ \frac{1}{2\sigma^2} \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2 + \frac{N}{2} \log(2\pi\sigma^2). \qquad (6.3)
\end{aligned}
$$

The negative log-posterior is partitioned in several distinct terms. The initial line showcases the negative log-prior with the total number of parameters M. It is scaled by the precision parameter λ and depends on the L_2 norm of the parameters. The second line corresponds to the negative log-likelihood that is scaled by the data variance σ^2 and determined by the L_2 norm of the residuals between observed and predicted values. In particular, the equation contains two logarithmic terms that are independent of both the parameter values and the residuals. The evidence has been neglected since it is not relevant to determine the MAP.

6.3.1 Estimating Local Uncertainty via Linearized Laplace Approximation

This section outlines the application of the Laplace approximation[1] *post-hoc* to pre-trained models, such as learned equations denoted by $\boldsymbol{f}_{\boldsymbol{\theta}}$, to capture its local uncertainty as illustrated in figure 6.2. In general, it approximates a probability distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ at its mode by matching a multivariate Gaussian $q(\boldsymbol{\theta} \mid \mathcal{D})$ as illustrated in figure 6.1 and given by

1: For a detailed derivation of the Laplace approximation, readers are directed to Chapter 4.4 of "Pattern recognition and machine learning" by Bishop[4]

$$
p(\boldsymbol{\theta} \mid \mathcal{D}) \approx q(\boldsymbol{\theta} \mid \mathcal{D}) = \mathcal{N}(\boldsymbol{\theta} \mid \tilde{\boldsymbol{\theta}}, \boldsymbol{A}^{-1})\,. \qquad (6.4)
$$

In this case, the posterior distribution is approximated at its mode $\tilde{\boldsymbol{\theta}}$ by a multivariate Gaussian with covariance $\boldsymbol{A}$. Therefore, a *second order Taylor series* at mode $\tilde{\boldsymbol{\theta}}$ is applied to the unnormalized log-posterior from equation (6.3)

second order Taylor series

$$-\mathcal{L}(\mathcal{D}, f_{\boldsymbol{\theta}}) \approx -\mathcal{L}(\mathcal{D}, f_{\tilde{\boldsymbol{\theta}}}) - \frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{A}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})\,. \qquad (6.5)$$

Its covariance is defined by the inverse curvature of the negative log-posterior

$$\begin{aligned} \boldsymbol{A} &= \nabla\nabla^\top \mathcal{L}(\mathcal{D}, f_{\boldsymbol{\theta}}) \mid_{\tilde{\boldsymbol{\theta}}} \\ &= -\nabla\nabla^\top \ln(p(\mathcal{D} \mid \boldsymbol{\theta})) \mid_{\tilde{\boldsymbol{\theta}}} -\nabla\nabla^\top \ln(p(\boldsymbol{\theta})) \mid_{\tilde{\boldsymbol{\theta}}}\;. \end{aligned} \qquad (6.6)$$

The first term in equation (6.6) is the Hessian of the negative log-likelihood $\boldsymbol{H}$. The second term corresponds to the prior. Equation (6.4) follows from taking the exponential of equation (6.5) and the corresponding Gaussian normalization. The predictive distribution of $f^*_{\boldsymbol{\theta}}$ at any test point $\boldsymbol{x}^*$ is given by integration over the weights as described in equation (2.3)

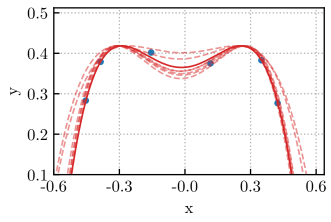


Figure 6.2: Illustration of local uncertainty given by measurement noise and uncertainty in parameter values of a plausible equation structure for the toy example from equation (E2). The dashed lines are possible samples of weight combinations within the same equation structure as the red line. The blue points represent the synthetic dataset. The shaded area reflects the local Laplace approximation.

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \mathcal{D}) = \int p(\boldsymbol{y}^* \mid f_{\boldsymbol{\theta}}(\boldsymbol{x}^*))\mathcal{N}(\boldsymbol{\theta} \mid \tilde{\boldsymbol{\theta}}, \boldsymbol{A}^{-1})\mathrm{d}\boldsymbol{\theta}\,. \qquad (6.7)$$

The integration is typically intractable. It can be approximated either by Monte-Carlo integration from parameter distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ or with a linearization at its mode $\tilde{\boldsymbol{\theta}}$

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}^*) \approx f_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^*) + \boldsymbol{J}^{*\top}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) \qquad (6.8)$$

with Jacobian $\boldsymbol{J}^* = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x}^*)|_{\tilde{\boldsymbol{\theta}}}$. This leads to a tractable predictive Gaussian distribution whose mean is given by equation $f_{\tilde{\boldsymbol{\theta}}}$ with parameters set to its mode $\tilde{\boldsymbol{\theta}}$

linearized Laplace approximation

$$\begin{aligned} p(\boldsymbol{y}^* \mid \boldsymbol{x}^*, \mathcal{D}) &\approx \mathcal{N}(\boldsymbol{y}^* \mid f_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^*), \boldsymbol{\Sigma}^*) & (6.9) \\ \boldsymbol{\Sigma}^* &= \boldsymbol{J}^{*\top}\boldsymbol{A}^{-1}\boldsymbol{J}^* + \sigma^2\boldsymbol{I}\,. & (6.10) \end{aligned}$$

The first term of the covariance $\boldsymbol{\Sigma}^*$ depends on the value of test point $\boldsymbol{x}^*$ due to the Jacobian $\boldsymbol{J}^*$ and describes the uncertainty of the equation structure that is represented by $\tilde{\boldsymbol{\theta}}$. The second term is determined by measurement uncertainty.

6.3.2 Estimating Global Uncertainty via Mixture of Laplace Approximations

We capture the global uncertainty of K plausible equations $f^k_{\tilde{\boldsymbol{\theta}}}$ with a mixture of Laplace approximations (MoLA) as described by Eschenhagen et al. [22] and illustrated in figure 6.3

[22] Eschenhagen, Daxberger, Hennig, and Kristiadi (2021), "Mixtures of Laplace Approximations for Improved Post-Hoc Uncertainty in Deep Learning"

$$p(\boldsymbol{y}^* \mid \boldsymbol{x}^*, \mathcal{D}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\boldsymbol{y}^* | f^k_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^*), \boldsymbol{\Sigma}^*_k) \tag{6.11}$$

The mixture coefficients π_k are chosen such that they reflect how plausible each equation is w.r.t. to the data. From a Bayesian perspective, the natural choice is the marginal likelihood, balancing accuracy and model complexity of the equation. It is expressed as

$$p(\mathcal{D} \mid f^k) = \int \mathrm{e}^{-\mathcal{L}(\mathcal{D}, f^k_{\boldsymbol{\theta}})} \mathrm{d}\boldsymbol{\theta}\,, \tag{6.12}$$

with the empirical risk $\mathcal{L}$ as detailed in equation (6.3). This integral is approximated using the results of equation (6.5) and depends on the negative log-posterior value at its mode $\tilde{\boldsymbol{\theta}}$ and the determinant of the curvature matrix

approximation of marginal likelihood

$$\begin{aligned} p(\mathcal{D} \mid f^k) &\approx \mathrm{e}^{-\mathcal{L}(\mathcal{D}, f_{\tilde{\boldsymbol{\theta}}})} \int \mathrm{e}^{-\frac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top A (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})} \mathrm{d}\boldsymbol{\theta} \\ &= \mathrm{e}^{-\mathcal{L}(\mathcal{D}, f^k_{\tilde{\boldsymbol{\theta}}})} \sqrt{\frac{(2\pi)^M}{\det \boldsymbol{A}_k}}\,. \end{aligned} \tag{6.13}$$

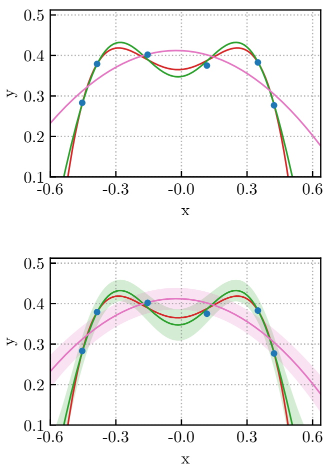


Figure 6.3: The two plots contrast three plausible equation structures (top plot) with their local Laplace approximations (bottom plot), illustrating the combination of global uncertainty and local uncertainty for the toy example from equation (E2).

Here, M_k represents the number of model parameters and $\boldsymbol{A}_k$ is the curvature matrix, with its determinant reflecting the width of the posterior landscape at its mode $\tilde{\boldsymbol{\theta}}_k$. Small Eigenvalues of $\boldsymbol{A}$ increase the marginal likelihood. They indicate directions in weight space that are not identified by data and thus represent flexibility of the equation in parameter space. The marginal likelihood can be decomposed in two contributions, the empirical loss favors accurate, yet simple equations and the determinant of the curvature favors flexible equations in parameter space.

The marginal likelihood is a central measure in Bayesian model selection and serves as a comparison criterion for different models. Usually, the hyperparameters are fine-tuned to maximize this marginal likelihood and thus control the selection of optimal models. Therefore, we choose the mixture coefficients π_k to be proportional to the respective normalized marginal likelihood

$$\pi_k = \Big(p(\mathcal{D} \mid f^k)\Big)^{1/N} / Z\,, \qquad Z = \sum_{k=1}^{K} \Big(p(\mathcal{D} \mid f^k)\Big)^{1/N} \tag{6.14}$$

This formulation deviates from traditional equation learning methods, which usually select a single specific equation. Instead, our multimodal mixture model captures several plausible equations, thereby accounting for local parametric uncertainties.

For *fast evaluation* we approximate the predictive distribution of the mixture of Laplace approximations with its first two moments similar to Lakshminarayanan et al. [46]

fast evaluation

[46] Lakshminarayanan, Pritzel, and Blundell (2016), "Simple and scalable predictive uncertainty estimation using deep ensembles"

$$\boldsymbol{m}^* = \sum_{k=0}^{K} \pi_k f^k_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^*) \tag{6.15}$$

$$\operatorname{diag}\boldsymbol{\Sigma}^* = \sum_{k=0}^{K} \pi_k (\operatorname{diag}\boldsymbol{\Sigma}^*_k + f^k_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^*)^2) - \boldsymbol{m}^{*2}\,. \tag{6.16}$$

The first moment, denoted as $\boldsymbol{m}^*$, is calculated as the weighted sum of the equations $f^k_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^*)$, according to equation (6.15). The second moment, denoted as $\operatorname{diag}\boldsymbol{\Sigma}^*_i$, builds upon the first moment. It is determined by the weighted sum of the diagonal elements of the covariance, plus a weighted sum of the squared equations, as shown in equation (6.16). This leads to the following Gaussian distribution

$$p(\boldsymbol{y}^* \mid \boldsymbol{x}^*, \mathcal{D}) = \mathcal{N}(\boldsymbol{y}^* \mid \boldsymbol{m}^*, \operatorname{diag}\boldsymbol{\Sigma}^*)\,. \tag{6.17}$$

We want to stress that the computational overhead is still small compared to neural networks, since just one forward pass and one backward pass of each equation is required to calculate the approximation. A visualization is shown in right panel of figure 6.4.

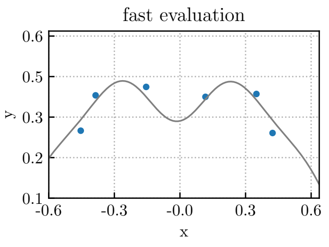


Figure 6.4: Illustration of the fast evaluation approximation for the toy example from equation (E2). The shaded area indicates 2σ standard deviation.

Remarks: The conditional mean of multimodal distribution can give a poor representation of the data. Especially, in extrapolation regions where the predictions of the underlying equation components strongly deviate. Depending on the application the conditional mode may be more meaningful. It would require numerical iteration since it has no analytic solution.

Our mixture of Laplace approximations strongly depends on the set of plausible equations by an equation learner. It inherits a bias towards structures of equations that occur more frequently. This might be wanted in the sense that the equation learner introduces a bias upon the equation structure.

6.4 Experiments

In this section, we investigate the predictive distribution of the MoLA with its local Laplace approximations. We highlight the importance of considering several plausible equations instead of one equation on the basis of two ambiguous toy datasets in section 6.4.1. In section 6.4.2 we study two real-world time series, which have been investigated in the course of structured uncertainty estimates by the *automatic statistician* [52].

[52] Lloyd, Duvenaud, Grosse, Tenenbaum, and Ghahramani (2014), "Automatic Construction and Natural-Language Description of Nonparametric Regression Models"

Training: We normalize input and output of the real-world datasets for training. We use the equation learner iEQL to retrieve plausible functions, which describe the datasets. As architecture, we use four hidden layers for the real-world datasets and three hidden layers for the toy datasets. Each hidden layer has $\{\cos, \exp\ , x^2, *\}$ as atomic units. Each atomic unit is applied four times in each layer. To account for the high frequency in the real-world datasets we use ten $\cos^*$ units with initial frequency[2] $f = 80$ on the normalized data instead of four cos units in each hidden layer. This mainly affects the initialization of the iEQL leading to a larger spectrum of frequencies due to the random initialization of the weight matrices. We prohibit combinations of cos(cos), exp(exp). The iEQL architecture for the real-world applications is illustrated in figure 6.5 More technical details are given in appendix section C.1.

2: This corresponds to a period of about one year in the Mauna dataset and about 100 days in the Airline dataset.

We refer to complexity in the framework of the iEQL. It measures the complexity of an equation by counting all active weights that are necessary to represent the equation.

Hyperparameters: We follow common practice to optimize the precision λ by maximizing the marginal likelihood. Therefore, we apply a grid search with $\lambda = 10^k$ where k is in the range from

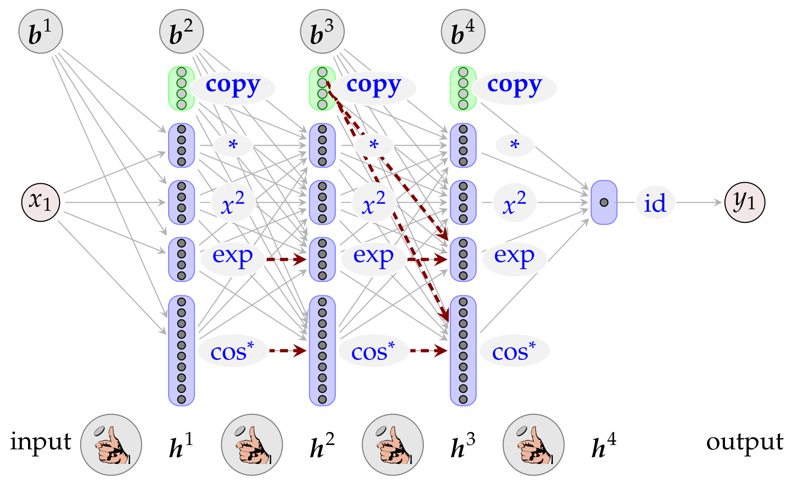
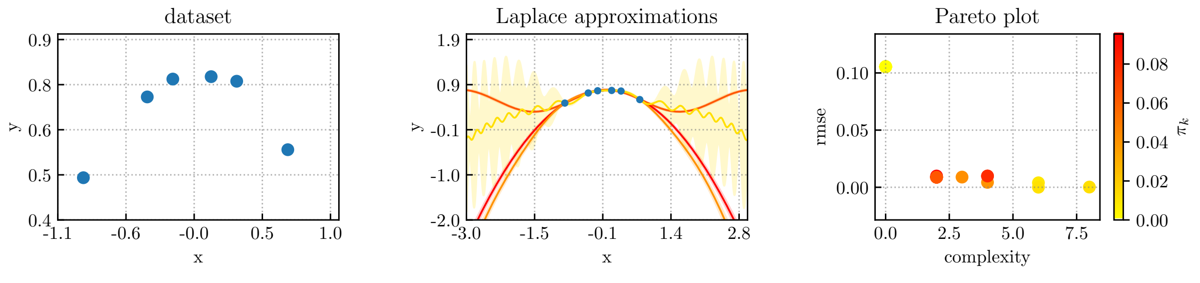
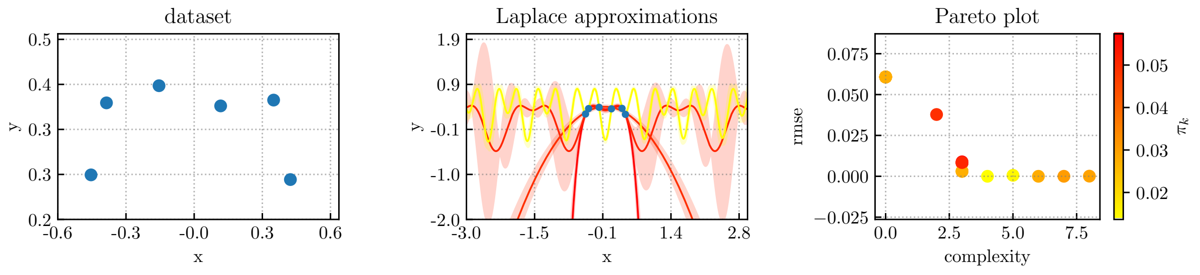


Figure 6.5: Illustration of the iEQL architecture for the real-world applications. Forbidden equation components cos(cos(·)) and exp(exp(·)) are marked in red.



(a) toy example equation (E1)



(b) toy example equation (E2)

Figure 6.6: Illustration of local uncertainty for the two toy examples from equation (E1) and (E2). The corresponding dataset is shown in the left panel. The middle panel shows the local Laplace approximations for four hand-picked equations from table 6.1. The shaded area indicates 2σ standard deviation. The color indicates the weighting of the mixture coefficients given by their normalized marginal likelihood. The color scheme is chosen such that it is aligned with the Pareto plot in the right panel, which shows root mean squared error (RMSE) on the train dataset over the complexity of each equation.

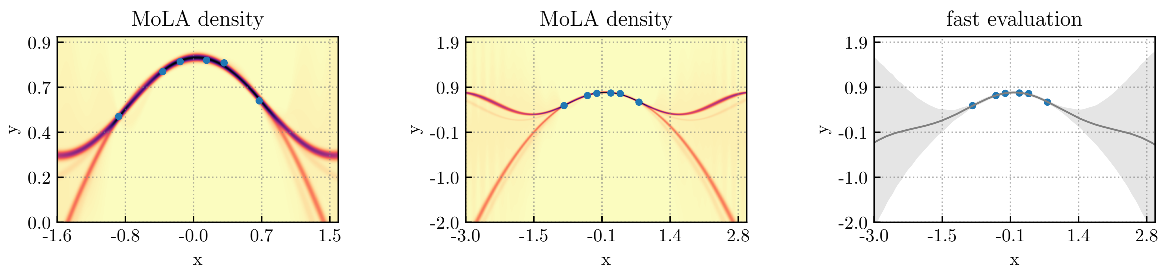
-2 to 2 with 100 equally spaced steps. The scale of the likelihood σ^2 is a model of measurement error. In technical applications, this parameter is usually known as part of the calibration of the measurement process and should then not be estimated. In situations where it is not known it can be estimated *post-hoc* empirically as $\sigma^2 = \sum_i^N (f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i)^2/N$, with the usual risk of model-overfitting.

Remarks: During our experiments we discovered that for deep neural networks and also for the considered equations the Hessian is not guaranteed to be positive-semi-definite (psd.) after convergence of the optimizer. This might be due to the use of the Adam [39] optimizer, which can converge to a saddle point with some directions with negative curvatures. Therefore, we approximate the Hessian with the generalized Gauss-Newton (GGN) matrix with backpack for pytorch [15], which is positive-semi-definite by construction. The use of a GGN approximated Hessian is motivated by the findings of Immer et al. [34], since we are using a linearized Laplace approximation.

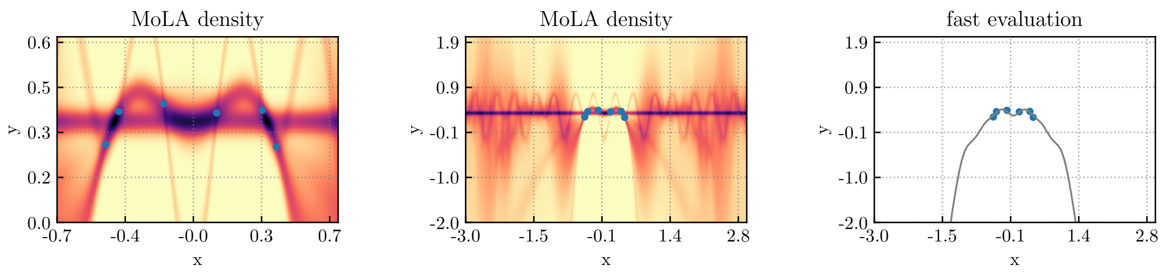
[39] Kingma and Ba (2014), "Adam: A method for stochastic optimization"

[15] Dangel, Kunstner, and Hennig (2020), "BackPACK: Packing more into Backprop"

[34] Immer, Korzepa, and Bauer (2021), "Improving predictions of Bayesian neural nets via local linearization"



(a) toy example equation (E1)



(b) toy example equation (E2)

Figure 6.7: Illustration of global uncertainty for the two toy examples from equation (E1) and (E2). The density distributions of the mixture of Laplace approximations are shown in the left and middle panel. The right panel shows the approximation for fast evaluation given by equation (6.17).

6.4.1 Illustrative toy examples

In order to highlight the importance of considering several plausible equations instead of one single equation as in conventional equation learning, we construct two ambiguous datasets in which , without further knowledge, polynomial and periodic equations are similarly plausible. Selecting one equation out of the others by chance means a loss of information about the data. The ground truth functions are chosen such, that they resemble a second order polynomial despite having a cosine structure with different measurement noise

$$y = 0.8\cos x + \epsilon\,, \qquad \epsilon \sim \mathcal{N}(0, 0.01^2) \tag{E1}$$

$$y = 0.8\cos x - 0.4 + \epsilon\,, \qquad \epsilon \sim \mathcal{N}(0, 0.03^2)\,. \tag{E2}$$

For each dataset six input values were uniformly sampled from $[-1, 1]$. The datasets are shown in the left panel of figure 6.6. We calculated 31 plausible equations for each dataset with different complexities with the iEQL. The right panel of figure 6.6 shows the corresponding Pareto plot. The iEQL found equations with complexities in the range of $[0, 8]$ for both toy examples. Their color indicates the weighting by the mixture coefficients π_k. Yellow indicates a small weighting and red indicates a strong weighting. Analytic expressions for four hand-picked equations are listed in table 6.1 along with their mixture coefficients π_k, root mean square error (RMSE) and complexity. Their local Laplace approximations are shown in the middle panel of figure 6.6.

Table 6.1: Hand-picked equations of different complexity for toy example E1 and E2 with root mean squared error (RMSE) and mixture coefficients π_k on the train dataset. Their local Laplace approximations are shown in the middle panel of figure 6.6.

dataset	π_k	rmse	complexity	equation
E1	0.095	0.0098	2	$y_0 = 0.79 - 0.34\,(1.05x_1 + 0.03)^2$
	0.063	0.0088	2	$y_0 = 0.56 - 0.24\cos(2.02x_1 + 3.16)$
	0.041	0.0043	4	$y_0 = -0.45\,(-0.98x_1 - 0.04)^2 + 0.02\cos(3.92x_1 - 2.2) + 0.81$
	0.012	0.00021	8	$y_0 = -0.15\,(0.85x_1 - 0.04)^2 - 0.02\cos(2.54x_1 + 3.08)$ $-0.04\cos(2.62x_1 - 3.2) - 0.07\cos\left(7.78\,(0.85x_1 - 0.04)^2 - 3.64\right) + 0.67$
E2	0.057	0.0088	3	$y_0 = 0.44 - 0.7\left(0.3 - 1.24\,(1.88x_1 + 0.06)^2\right)^2$
	0.051	0.0084	3	$y_0 = 0.45 - 0.45\,(0.5 - 0.98\cos(4.03x_1 + 0.13))^2$
	0.049	0.038	2	$y_0 = 0.43 - 0.52\,(-1.15x_1 - 0.05)^2$
	0.014	7.8e-06	4	$y_0 = 0.82 - 0.6\,(-0.01x_1 + 1.16\cos(7.54x_1 + 0.26) + 0.2)^2$

The first two equations of toy example (E1) are the two dominant modes given by a cos and a parabola equation. They can be clearly identified in the density distribution in the left and middle panel of figure 6.7. The two other equations are more accurate, but also more complex. Their weighting coefficient is smaller and thus they are hard to identify in the density plot, but their local Laplace approximation is shown in the middle panel.

The first three equations of toy example (E2) are the most dominant modes given by a fourth order polynomial, a squared cos and a parabola equation in descending order. The last hand-picked equation is orders of magnitude more accurate yet its mixture coefficient is four times smaller compared to the dominant modes. This can be related to larger Eigenvalues of its curvature matrix, meaning less flexibility in parameter space. Its local Laplace approximation is the yellow, high frequency line shown in the middle panel of figure 6.6.

6.4.2 Real-world time series

In the following section, we investigate two real-world time series datasets[3], which have been studied in the course of structured uncertainty estimates by the *automatic statistician* [20, 52, 77]. We present similar structured uncertainty estimates and provide interpretable, analytic expressions instead of non-parametric Gaussian processes for

Mauna Loa atmospheric CO_2 concentration (Mauna) recorded at the Mauna Loa observatory with 545 datapoints

Airline passenger data (Airline) of monthly totals of international airline passengers with 144 datapoints.

3: The datasets are downloaded from `https://github.com/ssydasheng/Neural-Kernel-Network`

[20] Duvenaud, Lloyd, Grosse, Tenenbaum, and Zoubin (2013), "Structure Discovery in Nonparametric Regression through Compositional Kernel Search"

[52] Lloyd, Duvenaud, Grosse, Tenenbaum, and Ghahramani (2014), "Automatic Construction and Natural-Language Description of Nonparametric Regression Models"

[77] Sun, Zhang, Wang, Zeng, Li, and Grosse (2018), "Differentiable compositional kernel learning for Gaussian processes"

Table 6.2: Hand-picked equations for real-world datasets Mauna and Airline with root means squared error (RMSE) and mixture coefficients π_k on the train dataset. Their local Laplace approximations are shown in the left and middle panel of figure 6.9.

dataset	π_k	rmse	equation
Mauna	0.029	0.4	$y_0 = 0.89x_1 + 0.42\,(0.52x_1 + 0.22)^2 - 0.15\cos(82.73x_1 - 3.49) + 0.23\left(-0.35x_1 + 0.85\,(-0.27x_1 + 0.66\cos(3.39x_1 + 5.09) - 0.28)^2 - 0.49\right)^2 - 0.1e^{0.41\cos(165.39x_1 - 5.67)} - 0.07$
	0.024	0.51	$y_0 = 0.8x_1 - 0.36\,(0.32 - 0.49\cos(82.69x_1 + 0.31))^2 + 0.09\cos(82.8x_1 - 1.18) + 0.01e^{2.42x_1} - 0.43e^{-1.9(-0.56x_1 - 0.43)^2} + 0.28$
	0.017	0.73	$y_0 = 0.83x_1 + 0.66\,(-0.41x_1 - 0.29)^2 + 0.29\,(0.69\cos(82.73x_1 - 14.03) + 0.08)^2 + 0.13e^{-0.99\cos(82.71x_1 + 2.99)} - 0.4$
Airline	0.039	8.7	$y_0 = 0.41x_1\,(0.23x_1 + 0.28) + 0.61x_1 + 0.4\,(0.86\,(-0.47\cos(65.09x_1 - 1.95) - 0.52)\,(0.77\cos(65.09x_1 - 1.95) + 0.09) + 0.98) \cdot (1.53\cos(21.78x_1 - 0.58) + 0.74\cos(22.06x_1 + 3.28) + 0.04) + 0.21e^{0.38x_1 - 1.92\cos(21.78x_1 - 0.58)} + 0.18e^{-2.6(1.02 - 1.92x_1)^2} - 0.63$
	0.027	18	$y_0 = 0.75x_1 + 0.1e^{0.64x_1 - 2.0\cos(21.75x_1 - 6.67)} - 0.28$

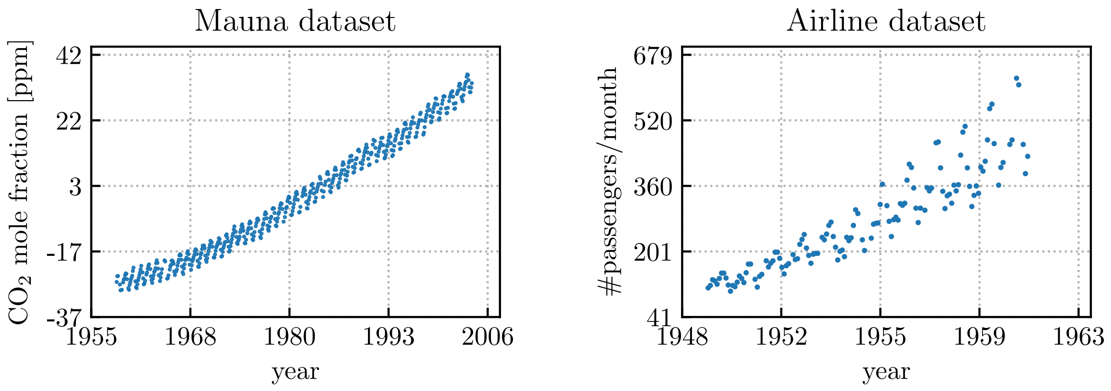
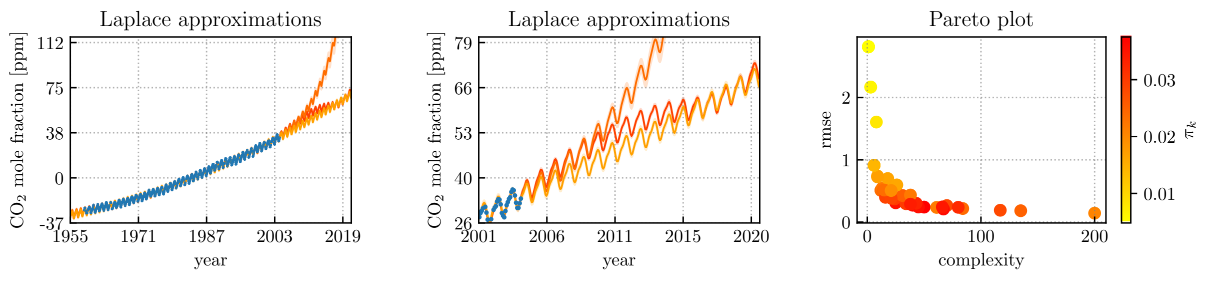


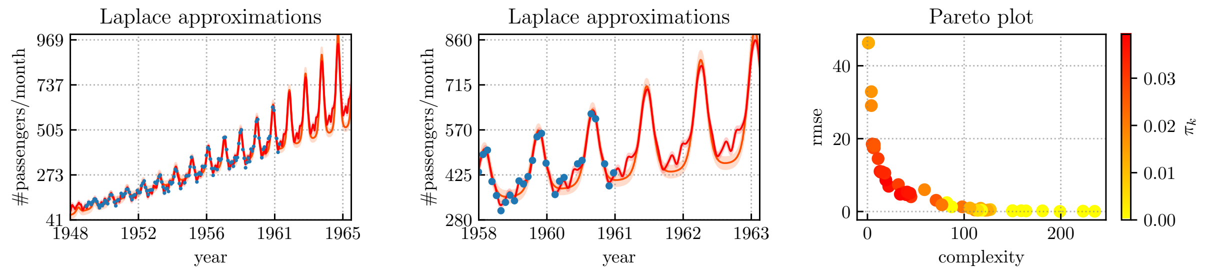
Figure 6.8: Real-world time series datasets: Mauna Loa atmosperic CO$_2$ concentration on the left hand side and on the right hand side Airline passenger data of monthly totals of international airline passengers.

We focus on extrapolation to examine our method's ability to discover the underlying structured uncertainty. We calculated 51 plausible equations for each dataset with the iEQL, which is shown in figure 6.5. The equations capture the underlying structure of the two datasets, which is illustrated in the left and middle panels of figures 6.9 and 6.10. Their complexity lies within [1 − 235] parameters. The mixture coefficients reliably prefer accurate and simple equations as shown in the Pareto plots in figure 6.9. Figure 6.10 shows the predictive distribution of the MoLa in the left panel and the middle panel shows a cutout area. The predictive distribution clearly indicates that the predictions of the equations diverge in the extrapolation area, as expected. The right panel shows the approximation of the predictive distribution for fast evaluation. It captures the underlying structure of the dataset and provides calibrated uncertainty estimates.

This is in contrast to the local Laplace approximations, which are



(a) Mauna dataset



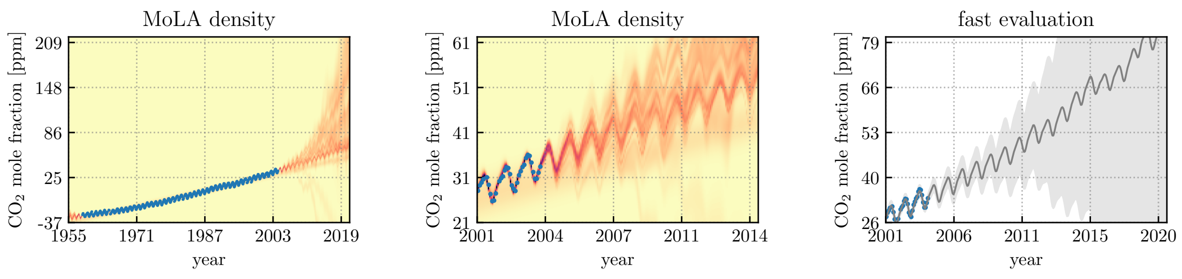
(b) Airline dataset

Figure 6.9: Illustration of local uncertainty for the Mauna and Airline dataset. The left andmiddle panel show the local Laplace approximations for hand-picked equations from table 6.2. The shaded area indicates 2σ standard deviation. The color indicates the weighting of the mixture coefficients given by their normalized marginal likelihood. The color scheme is chosen such that it is aligned with the Pareto plot in the right panel, which shows root mean squared error (RMSE) on the train dataset over the complexity of each equation.

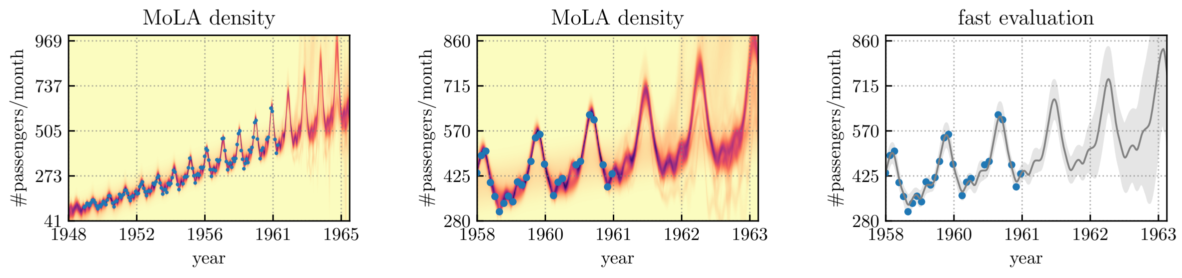
known to underestimate the uncertainty as shown in the left and middle panel of figure 6.9 for three hand-picked equations for the Mauna dataset and two hand-picked equations for the Airline dataset. Their corresponding mathematical expressions are shown in table 6.2.

We found that our method estimates the correct dominating frequency in all selected equations.
In the Mauna dataset the predictions deviate in the extrapolation region due to their differences in structure, which is an important motivation to capture *global uncertainty* with a mixture model. Especially, the second equation models the growth of the data with an exponentially growing contribution e^{ax_1}, whereas the first equation uses a fourth order polynomial and the third equation uses a parabola.
In the Airline dataset the structure of the second equation is very simple and does not capture higher frequencies. This leads to a larger uncertainty of its local Laplace approximation.

Figure 6.11, which shows Pareto plots of the root mean square error (RMSE) for the test dataset rather than the training dataset, provides a valuable insight into the Bayesian model selection process. The normalized marginal likelihood seems to be a promising measure to achieve a balance between model complexity and prediction accuracy. Moreover, the data confirm that the above-mentioned



(a) Mauna dataset



(b) Airline dataset

Figure 6.10: Illustration of global uncertainty for the Mauna and Airline dataset. The density distributions of the mixture of Laplace approximations are shown in the left and middle panel. The right panel shows the approximation for fast evaluation given by equation (6.17).

balance is maintained even when considering the RMSE on a test dataset that is not included in the normalized marginal likelihood calculation. These results suggest that normalized marginal likelihood could serve as a robust criterion for model selection in the context of equation learning.

In this section, we showed that our method captures the underlying structure of the datasets and provides structured uncertainty estimates. The individual local Laplace approximations underestimate the global uncertainty. Our approximation for fast evaluation provides reliable structured uncertainty, yet this has to be applied with caution since the conditional mean of a multimodal distributions can lead to a poor representation of the data.

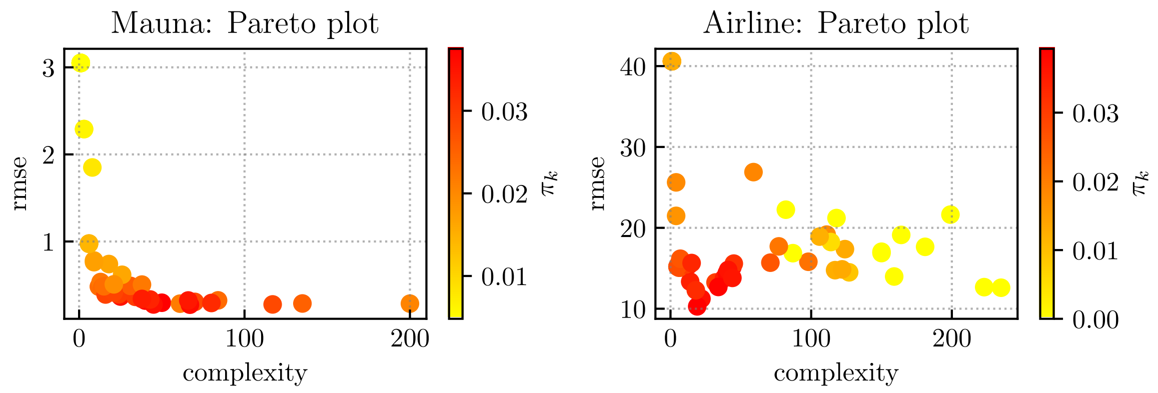


Figure 6.11: Pareto plots of Mauna (left) and the Airline (right) datasets. Instead of the RMSE on the train dataset, it displays the RMSE on the test dataset over complexity. The color indicates the weighting of the mixture coefficients given by their normalized marginal likelihood.

6.5 Conclusion

We introduced uncertainty in equation learning for any set of plausible equations found with an equation learner. We identified two components of uncertainty: *global uncertainty* given by the differences in structure of each equation and *local uncertainty* given by parametric uncertainty within one family of equations. We introduced a mixture of Laplace approximations to capture *global uncertainty* of several plausible equations. Each mixture component captures a different equation structure and the Laplace approximations capture local, parametric uncertainty within one family of equations. For computationally fast evaluation we proposed to match the first two moments of the mixture of Laplace approximations.

Part III

Conclusion & Future Directions

Conclusion & Future Directions | 7

The first section of this chapter provides a summary of the main contributions of this thesis. Subsequently, we present a perspective on potential future research topics that could broaden the application of equation learning neural networks.

7.1 Summary and Impact

This work contributed to the field of equation learning with neural networks by introducing new structures and methods to improve model expressivity and training stability. One of the main motivations was to scale equation learning neural networks to realistic conditions in science and engineering. Novel architectures and training techniques were developed to overcome the challenges of learning with functions that exhibit singularities, integrating expert knowledge into the learning process and using probabilistic optimization. The introduction of uncertainty quantification methods marked a significant step towards robust predictions in complex scientific and engineering tasks. The research demonstrated the improved ability of these networks to produce accurate and interpretable equations, with the added benefit of being able to quantify uncertainty within and across several plausible equations. This work provided critical insights to the three research questions Q1, Q2 and Q3 outlined in section 1.2.

research question 1

"How to broaden the expressivity of equation learning neural networks and train them efficiently?" Three approaches were presented in chapter 4 to improve the expressivity and stabilizing training of equation learning neural networks. Atomic functions with singularities, like logarithms and division, were incorporated across all layers, which posed a challenge in previous work due to instabilities during training. This incorporation broadened the network's ability to model complex equations. The integration of copy units enabled a unified computation of the Pareto-front avoiding extensive scans over network architectures with different numbers of hidden layers. Additionally, a probabilistic L_0 regularization method was employed to improve the overall training process. These developments collectively form a foundation for advancing equation learning to scale to real-world scenarios.

research question 2

“How to utilize domain knowledge to guide the search for better equations?” High expressivity inevitably requires an extensive search in the hypothesis space. Strategies to integrate domain and expert knowledge into both the training process and the structure of equation learning neural networks were explored in chapter 5. For this purpose, an informed equation learner iEQL was developed, which makes it possible to exclude combinations of functions and to prefer or reduce certain function types with the help of a domain-specific weighting scheme. The iEQL demonstrated success in identifying plausible and interpretable equations with strong predictive power across both artificial and real-world engineering datasets.

research question 3

“How to quantify uncertainty of a set of equations that was discovered by an equation learning neural network?” In chapter 6 we separated uncertainty into two contributions (i) global uncertainty, derived from structural differences between equations, and (ii) local uncertainty, related to parametric variability within an equation structure. It was possible to approximate these uncertainties using a mixture of Laplace approximations. Each mixture component captured a different equation structure, while the local Laplace approximation accounted for the corresponding parametric uncertainty. Its application was demonstrated using toy examples and two real-world datasets.

7.2 Outlook and Future Directions

This section describes possible research directions to extend the scope of equation-learning neural networks. The following ideas can be used to inspire new research projects.

Speed up Equation Learning: Equation learning neural networks are computationally intensive during training. As is common in deep learning, first-order optimization methods are used to optimize the parameters. Due to sparsity regularization and the diverse activation functions, training is computationally intensive compared to a fully connected neural network. However, in order to establish equation learning neural networks as an everyday tool for engineers and researchers, the calculation time must be significantly reduced. This section lists possible approaches to tackle this challenge.

Second order optimization Given the size of equation learning neural networks, the application of second-order optimization methods could speed up training. Current advances in deep learning provide tools and methods for second-order optimization. However, it will be challenging to combine these with an effective sparsity regularization.

Step-wise regularization In order to obtain a diverse Pareto front several networks with different regularization strengths must be trained. By gradually increasing the sparsity regularization, the number of individual trainings required can potentially be reduced to a minimum of one training run. In this way, equations of different sparsity and accuracy can be achieved within one training run. However, it is to be expected that the equations obtained will have a similar overall structure despite varying complexity. In order to obtain a diverse Pareto front within a training run, the optimization must offer the possibility of breaking out of an equation structure and explore other plausible structures.

Modeling heteroscedastic uncertainty Aleatoric uncertainty reflects the degree of noise present in observations. It can be categorized into homoscedastic (uniform noise across all inputs) and heteroscedastic (input-dependent noise) uncertainty. Chapter 6 presents an approach to capture structured uncertainty in equation learning based on the assumption of homoscedastic aleatoric uncertainty, characterized by a constant variance. Recent developments in Deep Learning explore methods for estimating both the variance and the mean in case of heteroscedastic uncertainty. Applying these techniques to equation learning could lead to more accurate and reliable predictions. These advances are promising for areas that require robust and reliable uncertainty estimates, such as active learning, safe reinforcement learning, and safety-critical systems.

Promoting interdisciplinary collaboration between mathematicians, computer scientists, engineers, and domain-specific scientists from other disciplines is crucial for the further development of equation learning. These collaborations ensure that the models are not only technically robust, but also practically relevant and geared to the requirements of real world applications. In order to fully utilize the strengths of equation learning neural networks, the focus should be on high-dimensional datasets containing numerous datapoints, where a non-trivial underlying equation is expected.

Part IV

Appendix

Additional Material for Chapter 4 A

A.1 Sparse Representations through L_0 Regularization

This section contains additional material on the differentiable L_0 regularization scheme of Louizos et al. [54]. It is applied in section 4.5 to optimize the objective loss function given by equation (4.10). Section A.1.1 describes the application of the reparametrization trick to the Bernoulli gates g. This method facilitates gradient backpropagation through stochastic variables. Section A.1.2 discusses the hard concrete distribution for the gates.

[54] Louizos, Ullrich, and Welling (2017), "Bayesian Compression for Deep Learning"

A.1.1 Reparametrization Trick

With the help of the reparametrization trick [40, 69] , gradient-based optimization can be applied to equation (4.12). Therefore, the gates z are hard-sigmoid rectifications $b(\cdot)$ of a continuous random variable s with variables ϕ

[40] Kingma and Welling (2013), "Auto-Encoding Variational Bayes"
[69] Rezende, Mohamed, and Wierstra (2014), "Stochastic backpropagation and approximate inference in deep generative models"

$$s \sim q(s \mid \phi)\,, \quad g = b(s)\,, \quad \text{with: } b(\cdot) = \min(1, \max(0, \cdot)). \tag{A.1}$$

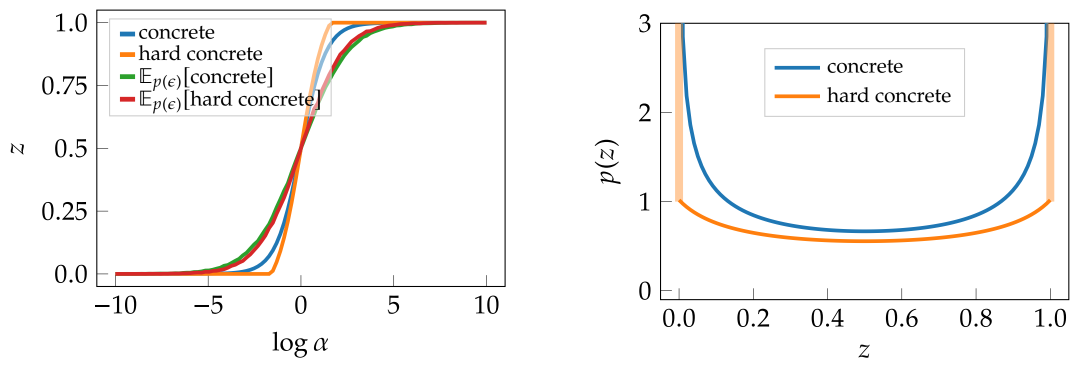
The expected loss is then

$$\begin{aligned}\mathcal{L} =& \mathbb{E}_{q(s|\phi)}\left[\frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{y}_i - \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{W} \odot \boldsymbol{b}(s)\|_2^2\right] \\ &+ \lambda \sum_{j\leq|W|}(1 - Q(s_j \leq 0 \mid \phi_j))\end{aligned} \tag{A.2}$$

with the cumulative distribution $Q(\cdot)$ of s. This penalizes the probability of a gate being non-zero. By selecting an appropriate continuous distribution $q(s)$, the reparametrization trick with parameter free noise distribution $p(\epsilon)$ and a differentiable transformation $f(\cdot)$ can be applied

$$\begin{aligned}\mathcal{L} =& \mathbb{E}_{p(\epsilon)}\left[\frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{y}_i - \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{W} \odot \boldsymbol{b}(f(\phi, \epsilon))\|_2^2\right] \\ &+ \lambda \sum_{j\leq|W|}(1 - Q(s_j \leq 0 \mid \phi_j))\end{aligned} \tag{A.3}$$

The expectation can be calculated with Monte Carlo approximation over the noise distribution $p(\epsilon)$. The final reparametrization is given by the hard concrete distribution outlined in the following.



(a) differentiable transformation (b) density plot

Figure A.1: The differentiable transformation of the binary concrete and hard binary concrete relaxation without noise as well as their expectations (equation (A.5) and equation (A.7) respectively) for 10000 samples are shown in (a). The probability density function for the concrete gate (equation (A.9)) and hard concrete gate (equation (A.11)). The delta-functions δ of the hard concrete distribution is indicated by the orange bars. All plots are produced with the hyperparameters mentioned in equation (A.16).

A.1.2 Hard Concrete Distribution

By choice, s is a binary concrete random variable [29, 58] distributed over $(0, 1)$ with probability density $q(s \mid \phi)$ and cumulative density $Q_\beta(s \mid \phi)$. The distribution has two parameters $(\log\alpha, \beta)$. Its location is denoted by $\log\alpha$ and the degree of approximation is controlled by β. A sampling method for the stretched version from a parameter free uniform distribution for the interval (γ, ζ)[1]is given by

[29] He, Zhang, Ren, and Sun (2016), "Identity Mappings in Deep Residual Networks"

[58] Maddison, Mnih, and Teh (2016), "The concrete distribution: A continuous relaxation of discrete random variables"

1: In order to ensure a "stretch" it is necessary that $\gamma < 0$ and $\zeta > 1$.

$$u \sim \mathcal{U}(0, 1) \tag{A.4}$$

$$s = \mathrm{Sigmoid}((\log u - \log(1-u) + \log\alpha)/\beta) \tag{A.5}$$

$$\bar{s} = s(\zeta - \gamma) + \gamma \tag{A.6}$$

$$g = b(\bar{s})\,. \tag{A.7}$$

The differentiable transformation s is provided by a sigmoid. Figure A.1 shows both differentiable transformations, the concrete and the hard concrete relaxation without noise as well as their expectations for 10000 samples on the left side and the corresponding probability distributions on the right side. The degree of approximation is determined by the temperature β. As the temperature approaches zero, the transformation converges towards the original Bernoulli distribution. However, this convergence is accompanied by a loss of differentiability. Further important properties of the binary concrete and the hard binary concrete random variables are given by

$$q_s(s \mid \phi) = \frac{\beta\alpha s^{-\beta-1}(1-s)^{-\beta-1}}{(\alpha s^{-\beta} + (1-s)^{-\beta/2})^2} \tag{A.8}$$

$$Q_s(s \mid \phi) = \text{Sigmoid}\left((\log s - \log(1-s))\beta - \log\alpha\right) \tag{A.9}$$

$$q_{\bar{s}}(\bar{s} \mid \phi) = \frac{1}{|\zeta-\gamma|} q_s\left(\frac{\bar{s}-\gamma}{\zeta-\gamma} \mid \phi\right) \tag{A.10}$$

$$Q_{\bar{s}}(\bar{s} \mid \phi) = Q_s\left(\frac{\bar{s}-\gamma}{\zeta-\gamma} \mid \phi\right) . \tag{A.11}$$

In the case, where $0 < \beta < 1$, the resulting probability density function concentrates its mass near the end points, which leads to a pronounced behavior of the distribution characteristics as shown in figure A.1(b). Finally, we obtain the probability distribution for the hard concrete sigmoid g

$$\begin{aligned} q(z \mid \phi) =& Q_{\bar{s}}(0 \mid \phi)\delta(z) + (1 - Q_{\bar{s}}(1 \mid \phi))\delta(z-1) \\ &+ (Q_{\bar{s}}(1 \mid \phi) - Q_{\bar{s}}(0 \mid \phi))\, q_{\bar{s}}(z \mid \bar{s} \in (0,1), \phi) \end{aligned} \tag{A.12}$$

The probability distribution of the hard concrete sigmoid (g) assigns weighted delta-peaks to its boundaries $\{0, 1\}$ because of the rectification. Thus, the complexity loss is defined by $1 - Q_{\bar{s}}(0, \phi)$ given by

$$\mathcal{L}_C = \text{Sigmoid}(\log\alpha - \beta\log\frac{-\gamma}{\zeta}) . \tag{A.13}$$

The corresponding expected dropout rate for $\beta = 0$ is given by

$$Q_{\bar{s}}(0, \phi) = \frac{1}{1+\alpha} \tag{A.14}$$

During the evaluation phase, the subsequent estimator masks the non-essential parameters using a hard concrete gate

$$\hat{z} = \min\left(1, \max\left(0, \text{Sigmoid}\left(\log\alpha\right)(\zeta-\gamma)+\gamma\right)\right) . \tag{A.15}$$

In our implementation we use the same hyperparameters as Louizos et.al. [55]

$$\zeta = 1.1, \qquad \gamma = -0.1, \qquad \beta = 2/3 . \tag{A.16}$$

Additional Material for Chapter 5 | B

B.1 Details on Training and Parameter Settings

All experiments are performed using an iEQL with four hidden layers. Each hidden layer has {cos, exp , log, $\sqrt{\ }$, x^2, $*$, /} as atomic units, and each atomic unit is applied four times in each layer. We prohibit the combinations cos(cos), cos(exp), exp(exp), log(log). The iEQL has thus 6405 learnable weights. Training is executed in two phases. In phase 1 we train for $T_1 = 2000$ epochs without regularization. This phase avoids bad initialization and assures that the model is close to a minimum when pruning starts. In phase 2 we train for $T_2 = 10000$ epochs with regularization strength λ. Just for the combustion engine dataset, which has about 5 to 6 times fewer datapoints, we increased the number of epochs by a factor of 8. We skipped train phase 1 for the ambiguous dataset since the iEQL is already sufficiently close to a minimum. After each epoch, an intrinsic penalty epoch is calculated with 100 randomly sampled datapoints from the test domain (without labels). The maximum desired output value is set to $B = 10$ with $\eta = 1$. We use the Adam optimizer [39] with learning rate 0.001, moving average $\beta_1 = 0.4$ and $\epsilon = 10^{-8}$ for numerical stability. The initial dropout rate[1] of the Bernoulli gates is set to 0.5 and the domain/bound penalty strength is set to $\delta = 1$. Further information on dropout rate is given in equation (A.14). Since the optimal regularization scale parameter λ is not known in advance, we train several models with different regularization strengths $\lambda = 10^k$, where k is in the range from -5.0 to 0.0 with 78 equally spaced steps. This results in 78 equations with different complexities and root mean squared errors (RMSE).

[39] Kingma and Ba (2014), "Adam: A method for stochastic optimization"

1: With the applied hard concrete approximation the dropout rate is $1/(1+\alpha)$ (see equation (A.14))

Baselines

We calculate the mean predictor (MP) on the train set and a multi-layer perceptron (MLP) with tanh activation functions and five hidden layers with 50 neurons each. It is trained with batch size 100 for 5000 epochs and the Adam optimizer with a learning rate of 0.001 and $\beta_1 = 0.9$. A grid search on the learning rate revealed that it is pretty robust in the range $[0.001, 0.0001]$. We select the best validation model to avoid overfitting.

We compare to EQL$^{\div}$, a state-of-the-art method from [72] with atomic unit types $\{\sin, \cos, *, \text{identity}\}$ and division in the final layer. It is sufficient to use the hyperparameters proposed in [72] since the datasets on which we compare have similar properties to the ones in [72]. So we applied Adam optimizer with learning rate 0.001 and $\epsilon = 10^{-4}$, mini-batch size 20, domain penalty $\eta = 10$ with $B = 10$, and 10 atomic units per type in each layer. The number of total epochs is given by $T = (L-1) \cdot 10000$, where L is the number of hidden layers. Just for the combustion engine dataset, which has about 5 to 6 times fewer datapoints, we had to increase the number of total epochs to $T = (L-1) \cdot 80000$. We perform model selection amongst the following parameters: regularization strengths $\lambda = 10^k$ where k is in the range from -6.0 to -3.5 with 26 equally spaced steps and $L \in \{2, 3, 4\}$. This results in 78 equations with different complexities and root mean squared errors (RMSE).

[14] Cranmer (2023), "Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl"

Another baseline is (PySR, [14]) a genetic algorithm for symbolic regression with hyperparameters shown in table B.1 for two different configurations (GA1, GA2). Both settings turned out to perform well on different datasets. We had to restrict the size of the datasets to 1000 datapoints in order to avoid exploding memory size. We do not compare to Eureqa, the current state-of-the-art tool for symbolic regression [19], since it has become proprietary and was merged into an online service.

[19] Dubčáková (2011), "Eureqa: software review"

Table B.1: Tuned hyperparameters for the genetic algorithm [14].

	GA1	GA2
niterations	40	10
npop	1000	1000
populations	60	30
binary operators	$\{\pm, *, /\}$	$\{\pm, *, /\}$
unary operators	$\{\cos, \exp, \log\}$	$\{\cos, x^2, \exp, \log\}$
maxsize	40	40
parsimony	0	0
warmupMaxsizeBy	0.5	0
useFrequency	False	True
annealing	False	False
optimizer algorithm	BFGS	BFGS
optimizer iterations	100	100
procs	30	30

For the real-world datasets a Gaussian Process (GP) is calculated with ASCMO [31], a standard tool from the engineering domain.

[31] Hoffmann, Schrott, Huber, and Kruse (2015), "Modellbasierte Methoden zur Applikation moderner Verbrennungsmotoren"

Power Loss of an Electric Machine Here, we present details on the preparation of training and test dataset. First, 20% of the whole dataset is used for testing and excluded from training. Then, the train domain is restricted to 80% of its range of operation ($[a, b] \rightarrow [0.8a, 0.8b]$ for each dimension). Rotor temperature T_{rot}

is not restricted, since it consists just of three different operation points. Further, 10% of the train dataset is used for validation. Restricting the train domain assures that the test dataset contains samples from an extrapolation domain as well as samples from the train domain.

B.1.1 A Closer Look at the Set of Plausible Equations of the Combustion Engine Dataset

We present the equations of median performance and the equations with the smallest number of parameters selected with the $\mathrm{V_{int}}$-S criterion for iEQL and $\mathrm{iEQL_{motor}}$. An overview of all four analyzed equations on the combustion engine dataset is shown in table B.2. Numbers were rounded to three figures and input and output dimensions were anonymized. The motor specific complexity factors lead to equations of mainly polynomial structure with higher degree than without motor specific complexity factors. The latter combine polynomials with simple nonlinear units like cos, exp, log or division units. Using more nonlinear units reduces the degree of the used polynomials.

iEQL	RMSE [Nm]	#parameters	equation
motor simple	6.16	15	(B.1)
motor median	3.17	32	(B.2)
plain simple	3.04	42	(B.3)
plain median	2.48	79	(B.4)

Table B.2: Overview of analysed iEQL expressions on the combustion engine dataset. The RMSE on the test dataset with the number of active parameters is shown.

Four analyzed equations on the combustion engine dataset

In this section, the equations from table B.2 are listed and their properties are commented on in bullet points.

iEQL$_{\mathbf{motor}}$ simple:

- compact notation of a polynomial of degree 8

$$
\begin{aligned}
y = 2.48x_2 + 0.66\,(0.3x_2 + 0.79)\,(-0.8x_1 + 1.86x_3 - 0.08) - 0.52\Big(-2.13x_3 + 0.57\,(0.05 - 1.19x_3)^2 \\
- 0.62\,(-0.89x_1 + 0.85\,(0.46x_1 - 0.32)\,(1.28x_3 + 0.34x_5 + 0.71) + 0.42)^2 + 0.56\Big)^2 + 0.94
\end{aligned}
\tag{B.1}
$$

iEQL$_{\textbf{motor}}$ **median:**

- compact notation of a polynomial of degree 7
- cos units in final layer with previous polynomial terms as arguments.

$$y = -0.3x_1 + 2.32x_2 + 0.27x_3 + 0.6c_2 + 0.3c_3 + 0.35\cos(1.12x_3 + 1.42c_2 + 0.59c_3 - 0.23) + 0.51 \tag{B.2}$$

substitutions:

$$c_1 = (0.05 - 0.69x_1)(1.0x_1 + 0.64)$$
$$c_2 = (-0.95x_2 - 0.69)\left(-1.06x_3 + 0.96(-0.21x_1 + 0.66x_3 - 1.01)^2 - 1.2\right)$$
$$\begin{aligned} c_3 = &\left(-0.74x_3 - 0.76\left(0.6x_1 + 0.73c_1 + 0.47(0.76x_2 + 0.19x_4 - 0.19x_5 - 0.6)^2 - 0.9\right)^2 + 1.36\right) \\ &(2.0x_3 + 0.88(-0.73x_1 - 1.17c_1 + 0.26)(0.6x_4 - 0.72x_5 - 0.46) - 0.31) \end{aligned}$$

iEQL **simple:**

- polynomial of degree 4 with simple cos and exp units
- cos units with previous polynomial terms as arguments.

$$\begin{aligned} y = {} & 1.49x_2 + 0.53(0.94x_2 + 0.8)(1.89x_3 - 0.29\cos(4.79x_3 + 2.83) + 0.46) \\ & + 0.81e^{-0.27x_1 + 0.18x_2} + 0.94e^{-0.32x_1 + 0.25x_2 + 0.55x_3} - 0.05c_1 \\ & + 0.22\cos(1.45x_3 - 1.02c_2 + 0.07) - 0.29\cos(0.66x_2 - 1.13c_2 - 0.74c_3 + 2.73) \\ & - 0.23\cos(-1.21x_1 + 1.99x_3 - 2.26(-0.94x_2 - 0.52)(0.97x_3 + 0.02) - 0.97c_3 + 3.71) \\ & - 1.71 \end{aligned} \tag{B.3}$$

substitutions:

$$c_1 = \cos(4.0x_1 + 2.38)$$
$$c_2 = \left(-0.49x_1 + 1.1x_3 + 0.76(1.02x_1 - 0.02)^2 - 0.85\right)^2$$
$$c_3 = (-0.41x_2 - 0.53x_3 - 0.29x_4 + 0.34x_5 + 0.59(0.07 - 0.97x_1)(0.28 - 1.02x_2) + 0.23c_1 + 0.6)^2$$

iEQL **median:**

- polynomial of degree 4 with simple cos, log and division units
- division units occur also in intermediate layers
- cos units with previous polynomial terms as arguments

$$\begin{aligned}
y = {} & 2.0x_2 + 0.6x_3 + 0.06x_4 - 0.05x_5 - 0.49\,(0.83x_2 + 0.58)\,(-0.81x_3 - 0.42) \\
& + \frac{0.39\,(0.99x_2 + 0.79)}{6.75\,(1.77c_1 - 0.75)^2 + 1.96} - 0.24\,(-0.77x_2 - 0.36c_1 - 0.42)\,\Big(1.11x_3 + 0.32c_2 \\
& \qquad - 0.89\,(-1.1x_3 + 0.73c_1 + 0.41)^2 - 0.47\cos(5.98x_3 + 3.27) + 0.09\Big) \\
& - 0.29\,\Big(-0.78x_2 + 0.35\,(1.77c_1 - 0.75)^2 + 0.35\cos(1.72x_2 - 2.6) - 0.67\Big)^2 - 0.28c_3 \\
& + 0.18\log(1.08 - 0.88x_1) - 0.2\cos(-1.21x_2 + 1.28c_3 + 4.61) \\
& - 0.15\cos(1.95x_2 + 1.58x_3 + 1.0c_4 + 3.51) \\
& + 0.18\cos(0.77x_1 - 1.71x_2 - 0.76c_4 + 0.71c_3 + 1.51) + 0.74 - \frac{0.39\,(0.85x_1 - 0.58)}{3.29 - 2.85x_2}
\end{aligned} \tag{B.4}$$

substitutions:

$$\begin{aligned}
c_1 &= \cos(2.07x_1 + 0.21x_2 - 0.02) \\
c_2 &= (0.33x_1 + 1.09c_1 + 0.51)\,(-0.74x_4 + 0.61x_5 + 1.02c_1 - 0.62) \\
c_3 &= \Big(0.29x_1 - 1.65x_3 - 0.09x_4 + 0.09x_5 - 0.27\,(-0.69x_1 - 0.14)^2 + 0.35\,(0.68x_1 - 1.27x_3 + 0.3)^2 + 0.4\Big)^2 \\
c_4 &= \left(-0.43x_4 + 0.36x_5 + 0.8 + \frac{0.86\,(-1.22x_2 - 0.94x_3 - 0.66)}{2.11x_1 + 2.35}\right) \\
&\quad (0.6x_2 + 1.28x_3 + 0.89x_4 - 1.18x_5 + 0.77c_2 - 0.05)
\end{aligned}$$

Additional Material for Chapter 6 | C

C.1 Details on Training and Parameter Settings

All experiments are performed using an iEQL architecture with three hidden layers for the real-world datasets and two hidden layers for the toy datasets. Each hidden layer has $\{\cos, \exp\ , x^2, *\}$ as atomic units and each atomic unit is applied four times in each layer. To account for the high frequency in the real-world datasets we use ten $\cos^*$ units with initial frequency[1] $f = 80$ on the normalized data instead of four cos units in each hidden layer.

1: This corresponds to a period of about one year in the Mauna dataset and about 100 days in the Airline dataset.

We prohibit the combinations cos(cos), exp(exp). We do not provide any domain expert knowledge and choose the domain specific complexity factors uniformly. For the given datasets penalty epochs are not necessary. We use the proposed optimizer setting with Adam and a learning rate $\alpha = 0.001$ without minibatches because of the high frequency of the data. The toy examples are trained for $T_1 = 60000$ iterations without regularization and for $T_2 = 100000$ iteration with regularization. The real-world dataset are trained for $T_1 = 200000$ iterations without regularization and for $T_2 = 600000$ iteration with regularization.

After convergence, we fine-tune the found equation with 40 steps using an L-BFGS optimizer with $\alpha = 1$ and a *strong wolfe* line search, since Adam is not guaranteed to converge.

In order to capture plausible equations of different complexity and accuracy, we train the iEQL with different regularization strengths $\lambda = 10^k$ where k is in the range from -3.0 to 0.0 with 31 equally spaced steps for the toy examples and the range from -5 to 0.0 with 51 equally spaced steps for the real-world datasets.

Each training for a single regularization strength was executed on a single CPU. A training lasts 9096 ± 2295 s for the Airline dataset, 11406 ± 1301 s for the Mauna dataset, 1578 ± 288 s for the toy dataset E1 and 1645 ± 180 s for the toy dataset E2.

Real-world dataset preparation: We normalize input and output of the real-world datasets for training. The datasets are split into 90% training and 10% testing.

Bibliography

[1] H. Akaike. “Information theory and an extension of the maximum likelihood principle”. *Selected papers of hirotugu akaike*. *Springer*, 1998.

[2] L. E. Ballentine. “The Statistical Interpretation of Quantum Mechanics”. *Reviews of Modern Physics* (1970).

[3] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo. “Neural Symbolic Regression that scales”. *International Conference on Machine Learning, ICML*. *PMLR*, 2021.

[4] C. M. Bishop. “Pattern recognition and machine learning”. *Springer*, 2006.

[5] A. Botev, H. Ritter, and D. Barber. “Practical Gauss-Newton optimisation for deep learning”. *International Conference on Machine Learning, ICML*. PMLR. 2017.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. “Language Models are Few-Shot Learners”. *Advances in Neural Information Processing Systems, Neurips*. *Curran Associates, Inc.*, 2020.

[7] S. L. Brunton, J. L. Proctor, and J. N. Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. *Proceedings of the National Academy of Sciences* (2016).

[8] J. S. Buchner, S. Boblest, P. Engel, A. Junginger, and H. Ulmer. “An Artificial-Intelligence-Based Method to Automatically Create Interpretable Models from Data Targeting Embedded Control Applications”. *IFAC* (2020). 21th IFAC World Congress.

[9] G. Camps-Valls, A. Gerhardus, U. Ninad, G. Varando, G. Martius, E. Balaguer-Ballester, R. Vinuesa, E. Diaz, L. Zanna, and J. Runge. “Discovering causal relations and equations from data”. *Physics Reports* (2023).

[10] K. P. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. “Data-driven discovery of coordinates and governing equations”. *Proceedings of the National Academy of Sciences of the United States of America* (2019).

[11] C. Cornelio, S. Dash, V. Austel, T. R. Josephson, J. Goncalves, K. L. Clarkson, N. Megiddo, B. E. Khadir, and L. Horesh. “Combining data and theory for derivable scientific discovery with AI-Descartes”. *Nature Communications* (2023).

[12] R. T. Cox. “Probability, frequency and reasonable expectation”. *American journal of physics* (1946).

[13] N. L. Cramer. “A Representation for the Adaptive Generation of Simple Sequential Programs”. *International Conference on Genetic Algorithms*. *L. Erlbaum Associates Inc.*, 1985.

[14] M. Cranmer. “Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl”. 2023. arXiv: `2305.01582 [astro-ph.IM]`.

[15] F. Dangel, F. Kunstner, and P. Hennig. “BackPACK: Packing more into Backprop”. *International Conference on Learning Representations, ICLR*. 2020.

[16] DataRobot Inc. “Eureqa as part of DataRobot’s service”. 2024. URL: https://www.datarobot.com/nutonian/.

[17] E. Daxberger, E. Nalisnick, J. U. Allingham, J. Antoran, and J. M. Hernandez-Lobato. “Bayesian Deep Learning via Subnetwork Inference”. *International Conference on Machine Learning, ICML. PMLR*, 2021.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologie. Association for Computational Linguistics*, 2019.

[19] R. Dubčáková. “Eureqa: software review”. *Genetic Programming and Evolvable Machines* (2011).

[20] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and G. Zoubin. “Structure Discovery in Nonparametric Regression through Compositional Kernel Search”. *International Conference on Machine Learning, ICML. PMLR*, 2013.

[21] K. Ellis, C. Wong, M. Nye, M. Sablé-Meyer, L. Morales, L. Hewitt, L. Cary, A. Solar-Lezama, and J. B. Tenenbaum. “DreamCoder: bootstrapping inductive program synthesis with wake-sleep library learning”. *International Conference on Programming Language Design and Implementation, SIGPLAN*. 2021.

[22] R. Eschenhagen, E. Daxberger, P. Hennig, and A. Kristiadi. “Mixtures of Laplace Approximations for Improved Post-Hoc Uncertainty in Deep Learning”. *CoRR* (2021).

[23] R. P. Feynman, R. B. Leighton, and M. Sands. “The feynman lectures on physics”. *American Journal of Physics* (1965).

[24] A. Y. Foong, Y. Li, J. M. Hernández-Lobato, and R. E. Turner. “’In-Between’Uncertainty in Bayesian Neural Networks”. *arXiv:1906.11537* (2019).

[25] C. E. García, D. M. Prett, and M. Morari. “Model predictive control: Theory and practice—A survey”. *Automatica* (1989).

[26] I. Goodfellow, Y. Bengio, and A. Courville. “Deep Learning”. http://www.deeplearningbook.org. *MIT Press*, 2016.

[27] F. Groß, M. Weigand, A. Gangwar, M. Werner, G. Schütz, E. J. Goering, C. H. Back, and J. Gräfe. “Imaging magnonic frequency multiplication in nanostructured antidot lattices”. *Phys. Rev. B* (1 2022).

[28] F. Groß, M. Zelent, A. Gangwar, S. Mamica, P. Gruszecki, M. Werner, G. Schütz, M. Weigand, E. J. Goering, C. H. Back, M. Krawczyk, and J. Gräfe. “Phase resolved observation of spin wave modes in antidot lattices”. *Applied Physics Letters* (2021).

[29] K. He, X. Zhang, S. Ren, and J. Sun. “Identity Mappings in Deep Residual Networks”. *European Conference on Computer Vision (ECCV 2016). Springer International Publishing*, 2016.

[30] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. *Neural Computation* (1997).

[31] S. Hoffmann, M. Schrott, T. Huber, and T. Kruse. “Modellbasierte Methoden zur Applikation moderner Verbrennungsmotoren”. *MTZ* (2015).

[32] J. J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” *Proceedings of the National Academy of Sciences* (1982).

[33] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. “Densely Connected Convolutional Networks”. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR* (2016).

[34] A. Immer, M. Korzepa, and M. Bauer. “Improving predictions of Bayesian neural nets via local linearization”. *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021.

[35] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. “Averaging weights leads to wider optima and better generalization”. *arXiv* (2018).

[36] E. T. Jaynes. “Probability theory: The logic of science”. *Cambridge university press*, 2003.

[37] Y. Jin, W. Fu, J. Kang, J. Guo, and J. Guo. “Bayesian symbolic regression”. *arXiv:1910.08892* (2019).

[38] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić. “Integration of neural network-based symbolic regression in deep learning for scientific discovery”. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[39] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. *arXiv:1412.6980* (2014).

[40] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. *CoRR* (2013).

[41] J. Koza. “On the programming of computers by means of natural selection, Genetic Programming, vol. 1”. 1992.

[42] A. Kristiadi, M. Hein, and P. Hennig. “Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks”. *International Conference on Machine Learning, ICML. PMLR*, 2020.

[43] A. Kristiadi, M. Hein, and P. Hennig. “Learnable Uncertainty under Laplace Approximations”. *arXiv* (2020).

[44] G. Kronberger, F. O. de Franca, B. Burlacu, C. Haider, and M. Kommenda. “Shape-Constrained Symbolic Regression—Improving Extrapolation with Prior Knowledge”. *Evolutionary Computation* (2021).

[45] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. “Grammar variational autoencoder”. *International Conference on Machine Learning, ICML*. JMLR. org. 2017.

[46] B. Lakshminarayanan, A. Pritzel, and C. Blundell. “Simple and scalable predictive uncertainty estimation using deep ensembles”. *arXiv:1612.01474* (2016).

[47] G. Lample and F. Charton. “Deep Learning For Symbolic Mathematics”. *International Conference on Learning Representations, ICLR*. 2020.

[48] W. B. Langdon, R. Poli, N. F. McPhee, and J. R. Koza. “Genetic programming: An introduction and tutorial, with a survey of techniques and applications”. *Computational intelligence: A compendium*. *Springer*, 2008.

[49] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. “Object Recognition with Gradient-Based Learning”. *Shape, Contour and Grouping in Computer Vision*. *Springer Berlin Heidelberg*, 1999.

[50] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. “Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks”. *International Conference on Machine Learning, ICML*. *PMLR*, 2019.

[51] S.-C. Lin, G. Martius, and M. Oettel. “Analytical classical density functionals from an equation learning network”. *The Journal of Chemical Physics* (2020).

[52] J. Lloyd, D. Duvenaud, R. Grosse, J. Tenenbaum, and Z. Ghahramani. “Automatic Construction and Natural-Language Description of Nonparametric Regression Models”. *Proceedings of the AAAI Conference on Artificial Intelligence* (2014).

[53] Z. Long, Y. Lu, and B. Dong. “PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network”. *Journal of Comp. Physics* (2019).

[54] C. Louizos, K. Ullrich, and M. Welling. “Bayesian Compression for Deep Learning”. *Advances in Neural Information Processing Systems, Neurips*. 2017.

[55] C. Louizos, M. Welling, and D. P. Kingma. “Learning Sparse Neural Networks through L_0 Regularization”. *arXiv* (2017).

[56] D. J. MacKay. “A practical Bayesian framework for backpropagation networks”. *Neural computation* (1992).

[57] D. J. MacKay, D. J. Mac Kay, et al. “Information theory, inference and learning algorithms”. *Cambridge university press*, 2003.

[58] C. J. Maddison, A. Mnih, and Y. W. Teh. “The concrete distribution: A continuous relaxation of discrete random variables”. *arXiv:1611.00712* (2016).

[59] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. “A simple baseline for bayesian uncertainty in deep learning”. *Advances in Neural Information Processing Systems, Neurips* (2019).

[60] J. Martens. “New Insights and Perspectives on the Natural Gradient Method”. *Journal of Machine Learning Research* (2020).

[61] J. Martens and R. Grosse. “Optimizing Neural Networks with Kronecker-factored Approximate Curvature”. *International Conference on Machine Learning, ICML. PMLR*, 2015.

[62] G. Martius and C. H. Lampert. “Extrapolation and learning equations”. *arXiv* (2016).

[63] T. McConaghy. “FFX: Fast, Scalable, Deterministic Symbolic Regression Technology”. *Genetic Programming Theory and Practice IX. Springer New York*, 2011.

[64] R. K. McRee. “Symbolic Regression Using Nearest Neighbor Indexing”. *Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO. *ACM*, 2010.

[65] T. J. Mitchell and J. J. Beauchamp. “Bayesian variable selection in linear regression”. *Journal of the American Statistical Association* (1988).

[66] T. Pearce, F. Leibfried, and A. Brintrup. “Uncertainty in neural networks: Approximately bayesian ensembling”. *International conference on artificial intelligence and statistics*. PMLR. 2020.

[67] B. K. Petersen, M. L. Larma, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim. “Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients”. *International Conference on Learning Representations, ICLR*. 2021.

[68] C. E. Rasmussen, C. K. Williams, et al. “Gaussian processes for machine learning”. *Springer*, 2006.

[69] D. J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic backpropagation and approximate inference in deep generative models”. *International Conference on Machine Learning, ICML. JMLR.org*, 2014.

[70] H. Ritter, A. Botev, and D. Barber. “A scalable Laplace approximation for neural networks”. *International Conference on Learning Representations, ICLR*. 2018.

[71] F. Rosenblatt. “Principles of neurodynamics. Percetrons and the theory of brain mchanisms”. *American Journal of Psychology* (1963).

[72] S. S. Sahoo, C. H. Lampert, and G. Martius. “Learning Equations for Extrapolation and Control”. *International Conference on Machine Learning, ICML. PMLR*, 2018.

[73] M. Schmidt and H. Lipson. “Distilling Free-Form Natural Laws from Experimental Data”. *Science* (2009).

[74] G. Schwarz et al. “Estimating the dimension of a model”. *The annals of statistics* (1978).

[75] D. P. Searson, D. E. Leahy, and M. J. Willis. “GPTIPS: an open source genetic programming toolbox for multigene symbolic regression”. *International Multiconference of Engineers and Computer scientists*. IMECS. 2010.

[76] T. Strauss, M. Werner, A. Junginger, M. Hanselmann, H. Ulmer, and K. Dormann. “Method and device for training and producing an artificial neural network”. Patent WO2020193481A1. 2020.

[77] S. Sun, G. Zhang, C. Wang, W. Zeng, J. Li, and R. Grosse. “Differentiable compositional kernel learning for Gaussian processes”. *International Conference on Machine Learning, ICML*. PMLR. 2018.

[78] R. Tibshirani. “Regression shrinkage and selection via the lasso”. *Journal of the Royal Statistical Society: Series B (Methodological)* (1996).

[79] L. Trujillo, L. Muñoz, E. Galván-López, and S. Silva. “neat Genetic Programming: Controlling bloat naturally”. *Information Sciences* (2016).

[80] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark. “AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity”. *Advances in Neural Information Processing Systems, Neurips*. *Curran Associates, Inc.*, 2020.

[81] S.-M. Udrescu and M. Tegmark. “AI Feynman: A physics-inspired method for symbolic regression”. *Science Advances* (2020).

[82] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. “Attention is All you Need”. *Advances in Neural Information Processing Systems, Neurips*. *Curran Associates, Inc.*, 2017.

[83] M. Virgolin and S. P. Pissis. “Symbolic Regression is NP-hard”. *Trans. Mach. Learn. Res.* (2022).

[84] M. Werner, A. Junginger, P. Hennig, and G. Martius. “Informed Equation Learning”. *arXiv: 2105.06331* (2021).

[85] M. Werner, A. Junginger, P. Hennig, and G. Martius. “Uncertainty in equation learning”. *Genetic and Evolutionary Computation Conference Companion, GECCO, Workshop Proceedings*. 2022.

[86] M. Werner, A. Junginger, P. Hennig, G. Martius, and M. Hein. “Apparatus and method for estimating uncertainties”. Patent DE102021124928A1. 2023.

[87] M. Werner, P. Margaretti, and A. Maciołek. “Drag Force for Asymmetrically Grafted Colloids in Polymer Solutions”. *Frontiers in Physics* (2019).

[88] A. G. Wilson, E. Gilboa, A. Nehorai, and J. P. Cunningham. “Fast Kernel Learning for Multidimensional Pattern Extrapolation”. *Advances in Neural Information Processing Systems, Neurips*. 2014.

[89] W. Zaremba, K. Kurach, and R. Fergus. “Learning to discover efficient mathematical identities”. *Advances in Neural Information Processing Systems, Neurips*. 2014.