Automated Knowledge Discovery Using Neural Networks

by

Maysum Panju

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Mathematics
in
Statistics and Actuarial Science

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Alejandro Murua Professor, Department of Mathematics and Statistics, Université de Montréal
Supervisor(s):	Ali Ghodsi Professor, Department of Statistics and Actuarial Science, University of Waterloo
Internal Members:	Matthias Schonlau Professor, Department of Statistics and Actuarial Science, University of Waterloo
	Yeying Zhu Assistant Professor, Department of Statistics and Actuarial Science, University of Waterloo
Internal-External Member:	Alex Wong Professor, Department of Systems Design Engineering, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The natural world is known to consistently abide by scientific laws that can be expressed concisely in mathematical terms, including differential equations. To understand the patterns that define these scientific laws, it is necessary to discover and solve these mathematical problems after making observations and collecting data on natural phenomena. While artificial neural networks are powerful black-box tools for automating tasks related to intelligence, the solutions we seek are related to the concise and interpretable form of symbolic mathematics.

In this work, we focus on the idea of a *symbolic function learner*, or SFL. A symbolic function learner can be any algorithm that is able to produce a symbolic mathematical expression that aims to optimize a given objective function. By choosing different objective functions, the SFL can be tuned to handle different learning tasks. We present a model for an SFL that is based on neural networks and can be trained using deep learning. We then use this SFL to approach the computational task of automating discovery of scientific knowledge in three ways.

We first apply our symbolic function learner as a tool for symbolic regression, a curve-fitting problem that has traditionally been approached using genetic evolution algorithms. We show that our SFL performs competitively in comparison to genetic algorithms and neural network regressors on a sample collection of regression instances. We also reframe the problem of learning differential equations as a task in symbolic regression, and use our SFL to rediscover some equations from classical physics from data.

We next present a machine-learning based method for solving differential equations symbolically. When neural networks are used to solve differential equations, they usually produce solutions in the form of black-box functions that are not directly mathematically interpretable. We introduce a method for generating symbolic expressions to solve differential equations while leveraging deep learning training methods. Unlike existing methods, our system does not require learning a language model over symbolic mathematics, making it scalable, compact, and easily adaptable for a variety of tasks and configurations. The system is designed to always return a valid symbolic formula, generating a useful approximation when an exact analytic solution to a differential equation is not or cannot be found. We demonstrate through examples the way our method can be applied on a number of differential equations that are rooted in the natural sciences, often obtaining symbolic approximations that are useful or insightful. Furthermore, we show how the system can be effortlessly generalized to find symbolic solutions to other mathematical tasks, including integration and functional equations.

We then introduce a novel method for discovering implicit relationships between variables in structured datasets in an unsupervised way. Rather than explicitly designating a causal relationship between input and output variables, our method finds mathematical relationships between variables without treating any variable as distinguished from any other. As a result, properties about the data itself can be discovered, rather than rules for predicting one variable from the others. We showcase examples of our method in the domain of geometry, demonstrating how we can re-discover famous geometric identities automatically from artificially generated data.

In total, this thesis aims to strengthen the connection between neural networks and problems in symbolic mathematics. Our proposed SFL is the main tool that we show can be applied to a variety of tasks, including but not limited to symbolic regression. We show how using this approach to symbolic function learning paves the way for future developments in automated scientific knowledge discovery.

Acknowledgements

First and foremost, thanks and praise belongs to God, without whom nothing would have been possible. I am deeply indebted to all of the sources of guidance, inspiration, and comfort that He placed and continues to place in my life.

I am incredibly grateful to my supervisor, Ali Ghodsi, for providing me with so much advice and direction throughout my graduate studies. Thank you for welcoming me as your student and for putting so much confidence in me throughout these years.

The research in this work greatly benefitted from useful discussions and group efforts with my peers and seniors, including Rui Qiao, Joseph Scott, Dhananjay Ashok, Sebastian Wetzel, Vijay Ganesh, Simon Fong, and Mojtaba Valipour. In particular, credit goes to Kourosh Parand for his help with the physics-related content of Chapter 4. Thank you all for sharing your knowledge with me and for the fruitful conversations.

This thesis, along with all of my post-secondary education leading up to it, owes a great deal to the amazing math teachers I had during my time at Richmond Hill High School. To Ms. Sinatra, Mr. Eschle, Mr. Brar, and Mr. Shim: thank you for inspiring me with a love and enthusiasm for mathematics that has both shaped my life and brought joy to it.

I would also like to gratefully acknowledge everyone who was rooting for me during my studies - it was so heartwarming to have all of you on my side. To my family, including my parents, my siblings, my niblings, and all of my wonderful in-laws: thank you for all of your love and constant encouragement; you really are The Best et al. Special mention goes to Fatima Janmohamed, Sakeena Panju, and Muhammed Abbas Panju for bearing with me since the very beginning. To my teachers, including Shaykh Salim Yusufali, Shaykh Jaffer Jaffer, and Dr. Hamidreza Mohebbi: thank you for helping me keep actively learning in areas beyond my thesis topic. To all of my friends, and in particular, Syed Kamil Kazmi, Sayyid Mohsin Jafri, Ali Reza Alibhai, Hussain Abbas, Miqdad Sopariwala, Kumail Janmohamed, Ali Al-Allo, and Sajjad Raza: it is a blessing and an honour to have companions like you.

A special thanks goes to my wonderful friend and mentor, Sajjad Rizvi [90], for tirelessly keeping me on track and motivated during my research and writing, carefully proofreading my thesis drafts, and all of the delightful and enriching conversations.

Finally, I would like to express my gratitude and appreciation for my wife, Sayyada Hussein, for always being a firm pillar of support and the best partner I could hope for. From the bottom of my heart, thank you.

Dedication

This thesis is dedicated to:

My father

for his guidance, encouragement, and unwavering confidence in me,

and:

My mother

for her warmth, inspiration, and desire to see me succeed.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

How can you remain patient with that which you do not fully understand?

Noble Qur'an, 18:68

It is a tremendous, and perhaps undeserved, blessing that the world we live in is governed at every level by consistent patterns. Rather than reducing to unpredictable chaos, the mechanics of the universe, ranging from the celestial motion of the stars to the behavioural tendencies of submolecular structures, seem to unwaveringly abide by a system of order and regularity. To understand the nature of the patterns behind these phenomena is to wield a knowledge about how different beings operate and interact, allowing us to make accurate predictions, reach intelligent decisions, and even ruminate on possible philosophies behind the workings of the universe.

One thing that helps enormously is the fact that the patterns underlying most of the natural world are quantifiable and, when presented correctly, understandable. "Mathematics is the language in which God has written the universe," Galileo is quoted as having said, and although this particular utterance may be apocryphal, the message has been repeatedly confirmed. Countless examples of natural phenomena have been successfully modelled using concise mathematical formulations represented as symbolic expressions. These expressions, as elegant as they are reliable, provide an empowering sense of insight on the inner workings of the world.

Although nature tends to abide by mathematical laws, it does not openly reveal them.

Behaviour, however regulated, must be observed and analysed in order to uncover the models underlying the patterns. The methods of developing these models may vary, but they all are based on the same learning process: gathering and processing observed data in order to obtaining new knowledge in the form of useful generalizations.

In the case of scientific laws, such as the mathematical equations that describe natural phenomena in the physical world, the techniques used for learning might traditionally involve a good amount of human ingenuity. Major breakthroughs often were not possible without the contributions of the likes of Newton or Einstein. Reaching a development might often require a creative mind, sharp insight, and sheer grit.

With the passage of time and the cumulative advancement of technology, tools are available now that previous generations of scientists could not have had access to. Machines now are capable of replacing humans in a variety of tasks, and not all of them are purely physical. The widespread rise of machine learning, artificial intelligence, and big-data computing, over the past few years, have made automation possible for a myriad of applications. This includes tasks that, until now, required an active input of human intelligence. It is only natural to question, then, whether algorithms might meet similar success when tasked with producing new results in science. Could the discovery of knowledge about the world around us be automated?

1.1 Three Tasks in Knowledge Discovery

Consider the example of the motion of an object of mass m falling to the ground from a height, influenced by both gravitational acceleration and air resistance. The object's velocity v at any given time t can be modelled by the mathematical equation

$$v(t) = \frac{m}{k}\left(g - Ce^{\frac{-k}{m}t}\right),$$

where k and C are constants that relate to the initial falling velocity and the strength of air resistance. This is an example of an explicit mathematical formulation of a purely physical phenomenon, where the velocity of the object depends on quantitative factors, including the point in time at which it is measured.

Or, for example, consider the dispersion of heat energy along a uniform rod along one spatial dimension. The heat energy, u, present at any given position x at a given time t is a solution to the concise partial differential equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2},$$

where α represents a measure of diffusivity. A particular solution can be specified by providing initial and boundary conditions for the given situation. In this case, we have a physical phenomenon modelled indirectly, where the amount of heat energy, as the quantity of interest, is defined as the solution of a differential equation.

As a final example, note that the angles and side lengths of a triangle are always intimately related in a holistic sense. The law of sines states that for any triangle, if we label the side lengths as a, b, and c, and the corresponding angles as A, B, and C, then the relationship

$$\frac{\sin(A)}{a} = \frac{\sin(B)}{b} = \frac{\sin(C)}{c}$$

will always hold. Here, we do not have a predictive model between input and output measurements, but a general relationship describing a structural property of geometric shapes.

In all of these examples, the behaviour of natural phenomena follows predictable patterns that can be spelled out explicitly using the language of mathematical equations. Such equations, and others like them, allow effective forecasting of how entities will interact and change. They also enable a deeper understanding of the nature of the phenomena, confirming trends that could previously only be guessed by looking directly at the data. For instance, the equation for the velocity of a falling object, as given above, shows us that the object will eventually converge towards an unchanging terminal velocity of mg/k.

In this work, we aim to present ways in which machine learning can be used for producing new knowledge about the natural world. In particular, we address three aspects pertaining to knowledge discovery from the perspective of machine learning, corresponding to the three example formulas shown above:

1. Quantitatively expressing causal relationships between measured observations, with minimal background knowledge. This relates to the task of **symbolic regression**.

2. Producing concise, interpretable solutions to satisfy known differential constraints, including the ones underlying many natural phenomena. This relates to the task of **symbolically solving differential equations**.

3. Discovering mathematically-expressible properties of structured objects in an unsupervised, data-driven way. This relates to the task of **unsupervised rule learning**.

1.2 Symbolic Function Learners

The main tool that plays a pivotal role in our approach to all three of the above tasks is what we call a Symbolic Function Learner, or SFL. In general, an SFL is any algorithm that aims to produce symbolic mathematical expressions that optimize a specified cost function. The key requirement of an SFL is that, given a class of symbolic functions F and a measure of cost on functions $L : F \to \mathbb{R}$, the SFL produces the minimizer

$$\arg\min_{f \in F} L(f),$$

or an approximation of it.

There are many ways to construct an SFL. Some of these approaches are outlined, along with a selection of related work, in Chapter 2. The traditional approach, dating back several decades, involves genetic evolution algorithms over the language of symbolic expressions. Most applications of symbolic function learning today are still based on these genetic methods.

More recently, work has begun on symbolic function learning using neural networks and deep learning. This area of research is still evolving and has room for much growth. In this thesis, we present a novel method for learning symbolic functions that is based on deep learning, which we describe in detail in Chapter 3. This SFL provides the main tool that we use for handling the automated knowledge discovery tasks that we approach throughout this work.

1.3 Contributions

In this thesis, we submit the following contributions:

1. We present a novel, deep-learning-based method for learning symbolic functions. Our proposed symbolic function learner, or SFL, is highly configurable, offering support for customized operations, and well-suited for a variety of tasks, as demonstrated throughout this work.

2. We show how our SFL naturally adapts to the task of symbolic regression, or learning mathematical formulas to fit datasets without pre-imposing constraints on the structure of the formula. We show through experiments that our method is able to perform competently when compared to existing methods for regression.

3. We apply symbolic regression, using our SFL, in the new context of producing differential equations to admit a given solution function. This is a relatively unstudied problem that has not been addressed using a symbolic deep learning approach before. We show that our method can recover known differential equations in physics, including special cases of Newton's second law.

4. We present a method for using a symbolic function learner that can be used to find symbolic solutions to ordinary and partial differential equations by means of a deep learning framework. This is an approach that has not received much attention other than by resorting to a large, trained language model.

5. We demonstrate how the symbolic function learner approach to solving differential equations can produce useful and insightful symbolic approximate solutions to differential equations that do not admit closed form solutions. This includes solutions to integrals and implicit functional equations.

6. We present a method for using a symbolic function learner to discover symbolic relationships between variables in a set, even when no specific regression structure is defined. This method can be used for performing an unsupervised discovery of mathematical properties that apply to a given set. We present examples of reproducing famous identities in geometry using this purely automated method.

7. We show how the above method can be used as an approach to symbolic regression, with the unique ability of learning symbolic equations that are implicitly defined in terms of an output variable.

We present these contributions, along with several diagrams and experimental results, in Chapters 3 to 5. Chapter 2 will focus on building up some background information on the tasks of symbolic regression and automated scientific discovery, along with references to several relevant works in these areas.

Chapter 2

Background and Related Work

Thus do We relate to you some accounts of what is past.

Noble Qur'an, 20:99

Our task of investigating methods of automated knowledge discovery is based on developing neural network-based methods for learning symbolic functions to describe patterns and relationships. In this chapter, we summarize some of the work that has been done towards this and related goals.

This chapter is divided into two parts. The first part is a discussion on the task of symbolic regression, as one of the most prevalent applications of symbolic function learning seen in literature. We also present a survey of algorithms specifically designed for this task. The second part focuses on works related to automated scientific discovery, particularly in the domain of physics, using both symbolic and non-symbolic methods. Together, these areas form the broader context in which the results of this work may be placed.

2.1 Symbolic Regression

The natural world consistently behaves in a predictable way, abiding by laws that are expressible using the language of mathematics. Discovering these laws involves extracting patterns from observations and identifying mathematical expressions to describe these

patterns. The task of symbolic regression, in particular, focuses on the problem of obtaining a closed-form symbolic mathematical expression to describe the relationship between specified predictor and response variables, where the mathematical expression is allowed to be flexible without being restricted to a particular structure or family.

Symbolic regression is, therefore, a much more difficult task than other kinds of regression, such as linear regression or multinomial regression, as the search space of candidate expressions is so much larger. Note, however, that even symbolic regression is not completely free of prior assumptions; it is still constrained by the choice of operations that are permitted in the sought equations.

Like all forms of regression, and as a special case of nonlinear regression, symbolic regression is a task in data analysis and is of interest in the field of statistics. Due to the highly computational nature involved in performing symbolic regression, the task is also connected with algorithms in computer science. Traditional approaches to symbolic regression use genetic evolution algorithms to navigate the large search space of symbolic functions. Recently, some research has begun approaching symbolic regression using methods in machine learning based on neural networks. We will present an overview of some work in these directions later in this chapter.

Symbolic regression can be understood through a variety of other names, including system identification [63], pattern recognition, and a special form of manifold learning. To illustrate the problem of symbolic regression in practice, we present the following example.

x_1	x_2	x_3	x_4	y
1.31	-0.25	7.01	-2.15	5.61
1.68	0.19	6.28	-3.62	14.50
2.15	0.04	13.75	4.58	22.48
1.56	0.01	-0.14	1.56	3.70
0.11	0.26	-0.31	-0.36	0.73

Table 2.1: Some hypothetical data values.

Table 2.1 contains a few rows from a sample dataset. The values of the target variable y are obtained through some mathematical formula $y = f(x_1, x_2, x_3, x_4)$. The goal of symbolic regression is to model the function f, given a table of values for each of the input variables and the target variable, such that $\mathbb{E}(y|x) = f(x)$. In this example, the predicted output might be $\mathbb{E}(y|x) = \sqrt{x_1 + x_2} + x_4^2$, which has a mean square error of approximately 0.003.

Since the model fitting the data now takes the form of a mathematical expression, it has the freedom to use not only sums and products, but also complex operations such as powers, trigonometric functions, and logarithms. Accordingly, the models obtained by symbolic regression can be much richer and more succinct than the models produced by neural networks. Furthermore, in contexts where symbolic regression applies, a trained model aims to learn the underlying concept precisely and exactly. The learned model is in the form of an equation where each variable, operation, and constant value can be meaningfully examined and interpreted in the context of the situation. The model is not chosen to simply fit the data, but to provide a mathematical explanation to reflect an understanding of the pattern.

The advantage of symbolic regression over other types of regression is clear: an investigator no longer needs to specify the structure of the regressing equation prior to performing the regression. However, this advantage comes with its own drawbacks, particularly in terms of computation. The extremely rich search space of all possible mathematical functions cannot feasibly be searched in its entirety. Thus, in order to discover the correct mathematical model to fit the data, clever algorithms must be employed to search this space intelligently.

Symbolic regression has tremendous utility in the natural sciences, particularly in physical dynamics, and hence has been studied to a great extent in many different ways and for many applications, including [16, 29, 85, 19, 17].

We now present some of the many approaches used to perform symbolic regression, along with some related research.

2.1.1 Genetic Algorithms

The traditional method taken for tackling symbolic regression computationally has largely been through methods that are based on the evolutionary algorithms. Here, the search process considers at each stage not a single candidate solution but a collection of candidates, known as a population. In each iteration, all candidates in the population are evaluated for how well they satisfy the objective, and those with higher scores are more likely to pass on to the next iteration. By introducing sources of random variation in each candidate solution, new candidates are generated, each with a possibility of being closer to the true target solution.

Evolutionary algorithms have been applied to symbolic regression with various levels of success; see, for example, [71, 9, 76, 109]. One evolution-based algorithm that has gained notable popularity is the EureQA method [93].

While differing on details, the general idea is to consider each candidate solution as a mathematical formula, with the evaluation score being a measure of how accurately the formula fits the given dataset. The target solution is the mathematical expression that describes the relationship of the variables in the dataset exactly, or as closely as possible. By going through the process of genetic evolution, the population of candidates is expected to grow better with each iteration, eventually producing at least one candidate formula that fits the dataset as desired.

The survey [79] describes the performance of various genetic programming approaches to symbolic regression in contrast non-symbolic regression methods used in machine learning. One method for evaluating the explainability of symbolic regression predictions is given in [73]. Various research studies have contributed discussion on the evaluation of different symbolic regression algorithms based on genetic programming [107, 116], as well as on the suitability of the benchmarks themselves [78].

Although the technique is now well-established, new variations on genetic methods for symbolic regression are still an area of active improvement. Interestingly, the task of symbolic regression has been used as a driving force behind advancements in genetic algorithms themselves. For example, [47] presents a method to modify genetic programming in order to handle asexual reproduction, claiming that this can allow rapid convergence to a global optimum while producing error that is less than what standard genetic programming would incur. [53] employs symbolic regression as a means to study population diversity dynamics of genetic algorithms. [41] suggests a way of improving genetic programming for symbolic regression by hybridizing with deterministic regression algorithms. [4] suggests an approach for imputing missing values in symbolic regression input using K-Nearest Neighbours. [23] tries to improve genetic programming by correlating residuals with variables in symbolic regression. [66] presents symbolic regression as one example to motivate a gravitational search algorithm, a close relative of genetic evolution algorithms, taking a population-based approach to automatically create computer programs that can be applied to tasks downstream.

2.1.2 Combinatorial Methods

Not all approaches to symbolic regression are based on genetic evolution. A few methods pursue the problem from the direction of combinatorics and discrete optimization. [98] is an example that seeks to perform symbolic regression by conducting a combinatorial search over symbolic equations, making use of tree-like representations of symbolic expressions. [26, 10, 49] take the approach of formulating the tree search as a mixed-integer programming

problem, and applying optimization techniques accordingly. While not related to symbolic regression directly, [33] uses Long Short-Term Memory networks (LSTMs) on grammar-based tree representations of symbolic equations for the task of simplifying mathematical expressions.

2.1.3 Sequence-Learning Methods

An interesting approach to symbolic regression is to frame the task as a sequence learning problem. In this formulation, the algorithm aims to output a sequence of tokens, where a token may be an operator, a variable, or a constant, in such a way that the resulting sequence can be parsed as the symbolic equation that fits the given input dataset. Often, these methods involve Recurrent Neural Networks (RNNs) such as LSTMs, in order to take advantage of their sequence-generating properties.

Within this framework, there are also different sub-approaches that have been taken. These include deep reinforcement learning, building a language model, and neural architecture search.

Deep reinforcement learning methods

A good example of a deep reinforcement learning method is described in [83] and [48]. Here, the authors use deep RNNs and deep reinforcement learning to learn symbolic regression as a sequence-learning problem. Another example of an approach based on reinforcement learning is [11], where the authors begin by randomly generating mathematical models and then use deep reinforcement learning methods to repeatedly improve their models.

Language-based models

A recent study [15] presents a novel method for handling symbolic regression as a machine translation task. Given an input dataset, the algorithm treats the input as a text string and passes it through a trained sequence-to-sequence LSTM to produce an output text string that is parsed as the desired symbolic expression. A current prototype of the method requires input data to be carefully structured along a mesh, and only produces the skeleton of the symbolic equation as output of the sequence model; the constants decorating the equation are learned separately as a subsequent optimization step.

In a similar vein, but outside the task of symbolic regression, [57] uses a language model for symbolic integration and solving first-order differential equations. The methodology involves producing large datasets using random function generators and symbolic computation engines and training a sequence-to-sequence neural network on this data until it is able to generalize beyond the capacity of the training set generator. An important contribution in this work is the algorithm for generating random functions as random symbolic parse trees.

Neural Architecture Search (NAS)

Closely related to the problem of symbolic regression is the task of neural architecture search, or NAS. Rather than finding the optimal structure and parameters of a mathematical formula to fit a dataset, NAS is focused on finding the optimal structure and parameters of a neural network that works best on the dataset. We mention this problem because of the tight similarity NAS shares with symbolic regression based on deep learning.

The seminal work of NAS sets up an RNN that outputs a sequential list of hyperparameters that define the architecture of a neural network [118]. This RNN is trained using the REINFORCE algorithm, where the reward used for training is the accuracy that the RNN's predicted architecture achieves, when trained on the dataset, against a held-out validation set. As the RNN training proceeds, it learns to produce successively better neural network architectures, because it is provided feedback that informs it which kind of architectures achieve higher accuracy scores than others. Since the seminal paper, NAS has been a hot area of active research: [112, 34, 88, 62] are some of the surveys of the advancements made in neural architecture search.

2.1.4 Grammar-Based Methods

These methods are closely related to sequence-based methods for symbolic regression, in that they treat symbolic expressions as sequences of tokens that are generated and later on interpreted as mathematical objects that can be evaluated to produce values. In contrast with the sequence-based methods, grammar-based methods exploit the fact that symbolic mathematical expressions exhibit a tree-like structure that can be associated with a grammar on symbols. In this way, grammar-based methods are also similar to combinatorial methods.

These methods use the grammar of symbolic mathematics to generate mathematical expressions as candidates for functions to fit the provided dataset. An example of this

approach is given in [18], which places a lot of focus on the context-free grammar of generating mathematical formulas. In another example [58], the researchers accompany the symbolic mathematics grammar rules with a Monte Carlo Tree Search, and also make use of some asymptotic properties as semantic priors of the function to be learned.

2.1.5 Neural Network-Based Methods

A more direct approach to symbolic regression takes advantage of the prevalence of deep learning models and tries to frame neural networks as a special case of a symbolic regression problem. The deep learning approach was initiated by Martius and Lampert in 2016 in a model called EQL (Equation Learner) [69], and extended by a team including the original authors in 2018 to a model called EQL$^{\div}$ [92].

The idea behind this method involves creating a type of node that allows operations such as multiplication and trigonometric functions, and building a network out of such nodes. The network is then trained in a manner similar to traditional neural networks with a regularization term in the cost function to encourage sparsity within the network edges. The resulting model is interpreted as a mathematical formula, the output of the symbolic regression.

There have been several approaches based on modifications and improvements to the basic method outlined by EQL. For example, [22] uses a similar framework but with an additional application of some methods from statistics. Some papers have used EQL as a starting point for downstream tasks, such as [50], which presents some interesting applications in image comprehension and learning physics.

[114] presents an interesting method that turns symbolic regression into a computer vision problem, training an encoder-decoder model based on a ResNet and a CNN on images of the data to try and learn operators in the symbolic parse tree.

2.1.6 Other Methods for Symbolic Regression

There are various other ways to approach symbolic regression that do not fit in the categories described so far. The method given by [21] introduces an algebraic method called Elite Bases Regression, which generates and updates a set of candidate basis functions that contain the regressor function in their span. [91] poses symbolic regression as a straight-line program and tackles it using an ant colony optimization approach; [65] takes a related but different direction with a particle swarm optimization approach. [102] presents a technique

that involves generating regressor functions purely at random, arguing that this method may be more robust than genetic models while achieving comparable performance in some situations.

2.1.7 Applications of Symbolic Regression

Symbolic Regression is very much inspired by, and hence useful for, research in physics and the natural sciences [3]. Its application, however, has not been limited to these areas alone. Some of the numerous applications that have benefited from models based on symbolic regression include:

- Storm surge forecasts [39]
- Hospitality research [103]
- Shear-strength of concrete walls [37]
- Solar radiation [25]
- Symbolic reward functions for deep reinforcement learning [95]
- Improving land cover classification [12]
- Hot water boiler modelling and identification [99]
- Property valuations [74]
- Star formation rate density in cosmology and astrophysics [108]
- COVID-19 epidemic case curves [5]
- Cleaning processes in the food industry [32]
- Effect of tax on profit shifting of multinational corporations worldwide [36]
- Semantic similarity between textual expressions [68]
- Wind speed forecasting [1]

2.2 Machine Learning for Scientific Discovery

As mentioned in Chapter 1, many behaviours observed in the natural world follow regular, mathematically-explainable patterns. This is particularly common in the realm of physics, where many problems revolve around motion and physical properties changing over time. For example, the position at time t of a particle of mass m moving along a one-dimensional damped oscillator is given by

$$x(t) = Ae^{\frac{-\alpha}{2m}t} \cos\left(\sqrt{\frac{k}{m} - \frac{\alpha^2}{4m^2}}t + \phi_0\right),$$

where A, α, and k are constants pertaining to the initial conditions of the specific instance.

Since so many of these problems involve quantities changing over time, many physical phenomena present themselves as suitable for representation in the form of differential equations. A differential equation is a special form of functional equation which defines a function in terms of how it is related with its derivatives. In general, a differential equation satisfied by the function y over the variables $x_1, \ldots, x_d$ can take the form

$$g\left(x_1, x_2, \ldots, x_d, y, \frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_d}, \frac{\partial^2 y}{\partial x_1^2}, \frac{\partial^2 y}{\partial x_1 \partial x_2}, \ldots\right) = 0.$$

We say a differential equation is an ordinary differential equation (or ODE) if it is in terms of one independent variable (and hence $d = 1$, and a partial differential equation (PDE) if there are multiple independent variables (when $d > 1$).

In the case of the damped oscillator example above, the position function is the solution to the differential equation

$$mx'' + \alpha x' + kx = 0,$$

which is a statement of Newton's second law applied to this situation.

Understanding the laws of physics allows us to improve the way we comprehend and interact with the world around us. While traditional methods of discovering the equations underlying physical properties and rules may have relied on insight and creativity, modern developments allow us to additionally harness the power of high-speed computing available to us. As a result, there have been many works of research around the topic of data-driven discovery of equations and formulas for use in the natural sciences [111].

In the following sections, we present some of the developments related to the advancement of machine learning for scientific discovery. Due to the nature of the task, many of these results overlap with the theme of symbolic regression, although the ones presented below have a particular focus on studying and learning the laws of nature.

2.2.1 Discovery of Natural Laws

With the advent of deep learning, artificial neural networks have become popular tools for modelling all kinds of natural applications, including the laws of science. SciNet [42] is an example of an approach that uses neural networks for recovering the laws of physics, including the heliocentric model of the solar system. While models based on neural networks can achieve high accuracy in fitness measures, they are not immediately interpretable, although [35] offers some insight on translating machine-learned physics to a more human-readable format.

Other approaches aim towards finding expressions in symbolic mathematics for physical phenomena directly. Publications such as [28, 110, 27, 24] all provide methods based on deep learning for obtaining symbolic models for learning physics directly. In particular, there is active research in unsupervised learning of the laws of natural science [44, 113, 38]. Generally, these approaches are suited to learning rules to described specific situations in physics, and make use of custom neural network architectures to achieve this goal, including graph neural networks and Siamese networks.

The AI Feynman model [106] presents an interesting approach to symbolic regression that capitalizes on using several heuristics based on generally recurring patterns in symbolic formulas describing natural phenomena. The paper employs a method of artificially generating data points using a neural network, which allows the correct formulas to be produced with greater likelihood. The method in this paper was able to achieve tremendous success in predicting many scientific formulas for natural phenomena. Its main restrictions are its reliance on heuristics and its inability to discover constant values, a common difficulty in symbolic regression algorithms [51]. The algorithm received some improvements in its successor paper, AI Feynman 2.0 [105], which focuses on finding symbolic expressions along a Pareto front of complexity and accuracy.

Not all data-driven methods for scientific discovery depend on deep learning, however. There are many examples of work that use standard methods in optimization and regression for learning natural laws. Some approaches include sparse nonlinear regression [43, 19], genetic evolution algorithms [97], and properties of non-parametric machine learning [61].

One direction for adapting symbolic regression to accommodate physics applications is the incorporation of background knowledge as auxiliary information provided to the learning task. This is the approach taken by [94, 8], which use logic solvers as a means of improving a symbolic regressor using background knowledge framed as statements in logic. A similar approach, without the logic solvers, is used in [54].

2.2.2 Learning Differential Equations

There are several interesting works for data-driven discovery of differential equations, usually falling into one of two categories: techniques based on sparse regression, such as [14, 45, 20, 117], and techniques that use neural networks to model the differential equations, such as [86, 84]. These latter works use black-box neural networks, rather than symbolic expressions, to represent the learned differential equations.

There are some approaches in symbolic differential equation discovery, such as a specialized deep learning model for the task presented in [64], and the method based on discovering variational laws given in [40]. The paper [70] presents a novel technique for learning systems of symbolic partial differential equations using a genetic approach similar to symbolic regression, rather than neural networks.

2.2.3 Solving Differential Equations

There are countless papers describing methods and applications of using neural networks for solving differential equations. Some of the interesting contributions made using this approach include [72, 55, 81, 13, 67, 87]. A genetic-based approach that makes use of neural networks is [104]. Primarily, these methods are based on numerical evaluations, rather than producing symbolic solutions to differential equations. Approaches based on neural programming are still largely under development [7].

The main work that uses deep learning for symbolic mathematics is given in [57]. The authors use a transformer network and an intelligently constructed training dataset to learn a language model over the language of symbolic mathematics. They treat differential equations and integration problems as input to a sequence-to-sequence network, trusting the language model to have learned the rules of mathematics after observing tens of millions of valid examples. This approach has worked so well that they are able to integrate expressions that even Mathematica cannot.

Although the results are remarkably impressive, they depend upon an extremely costly training procedure that cannot scale well to new configurations, for example, involving a different set of operations than the ones used in the training set. On a deeper level, the model has faced criticism for issues ranging from the artificiality of its testing to the fact that it "has no understanding of the significance of an integral or a derivative or even a function or a number" [30].

As pointed out in [30], this model is "like the worst possible student in a calculus class": it is able to work with mathematical symbols only because of an abundance of repeated

exposure, and not because of any inherent understanding or appreciation for any of the concepts actually represented.

In contrast, the model that we propose in Chapter 4 is like a student that has not studied the techniques used in the course, but tackles each calculus problem as a brand-new challenge, discovering and improving solutions in a trial-and-error based method. It may not be the best student, but it is certainly more dedicated and adaptable than the worst.

2.2.4 Neural Programming

Closely related to symbolic regression and symbolic function learning is the task of neural programming, which involves using neural networks to learn programs, mathematics and logic from data [6].

The idea behind learning operators is the task of learning a mapping between two infinite-dimensional spaces from a finite collection of data. The spaces here represent two classes of functions over a given domain in $\mathbb{R}^n$, which are the intended input and output of the operator. In this framework, we would not be learning the solution to a single differential equation, for example, but an entire family of differential equations: the input space for the operator would contain the differential equations and the output space would contain the solutions.

A neural operator is an approach to operator learning that uses a neural network to model the operator. In particular, the updates on hidden values between layers of the network involve non-linear functions that can take on different transformations ranging from arithmetic operators to integration and Fourier operators.

Instances of neural programming include [60], which used an implementation of graph kernel networks to approach the task of solving partial differential equations, and [59], which employs Fourier operators in each neural unit. Another example of neural programming is [6], which aims to supplement a Tree-LSTM model for symbolic function learning with additional stacks for storing memory.

The SFL presented in this work relates closely with the goals of neural programming, but presents an entirely novel approach by employing learnable gating operations to select operators within each layer. This will be explained in more detail in Chapter 3.

2.3 Summary

Symbolic regression is not a new problem, and there is a rich and active body of research dedicated to advancing methods for addressing this task. While most of these methods are traditionally related to genetic programming algorithms, approaches based on neural networks and deep learning have begun to gain traction and attention. The symbolic function learner presented in the next chapter aims to contribute to this evolving space.

There is also an active field of research in automating scientific discovery, including the study of differential equations. These methods may include approaches based on neural networks, but there seems to be a lack of attention to the potential of using deep learning to solve differential equations symbolically. The contributions presented in this thesis offer a method to fill this gap.

Chapter 3

A Neural Method for Symbolic Function Learning

> *Untie the knot from my tongue, that they may understand my speech.*
>
> Noble Qur'an, 20:27-28

In this chapter, we present a novel method for approaching symbolic regression using a deep learning approach. Our symbolic function learner, or SFL, provides a neural-based architecture that can be used to learn a symbolic function to optimize a given objective value. As a result, the SFL is capable of producing models in the form of mathematical expressions. These expressions have the advantage of being concise and directly interpretable to human readers, unlike the black-box models given by other regression methods such as standard neural networks.

In the task of symbolic regression, which was described in detail in Chapter 2, the objective value to minimize is the residual error between the true output values and values predicted by the fitted symbolic expressions. We demonstrate that our method outperforms conventional regression methods in specified experiments. We also apply our method to the task of learning symbolic differential equations, introducing a novel way for using symbolic function learners in a practical problem of scientific discovery.

3.1 Method

We propose a model for a multivariate Symbolic Function Learner (SFL), an algorithm that generates symbolic mathematical expressions that minimize a given cost function. Any algorithm designed with this generic goal is perfectly suited to handle the task of symbolic regression, where the cost function would correspond to the residual error of fit between the true and predicted function output values. In fact, many symbolic regression algorithms can be reframed as generic symbolic function learners with a little modification. In this section, we describe our approach in detail.

3.1.1 Expressions as Parse Trees

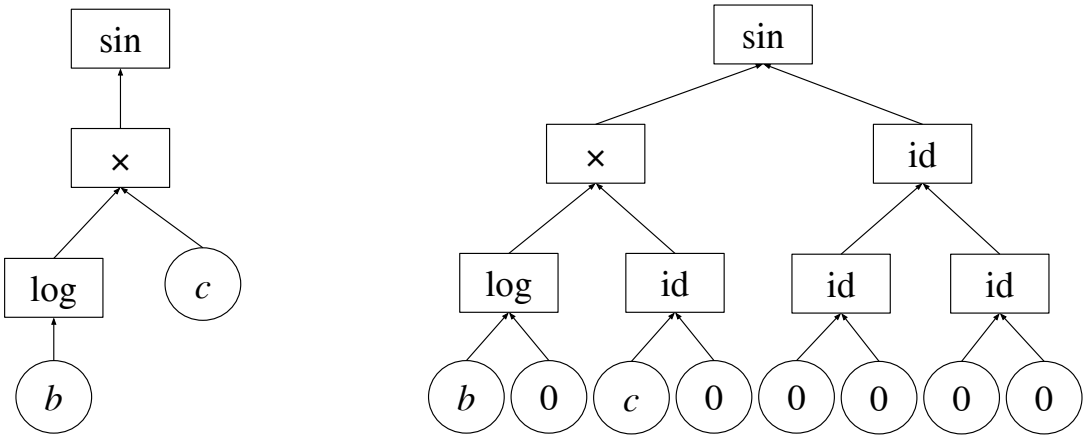

Figure 3.1: **Left:** A symbolic parse tree encoding the expression $\sin(\log(b) \times c)$ Leaf nodes represent variables and constants, and interior nodes represent operators. **Right:** the same expression represented using a parse tree that is balanced and properly binary, where the unary operators sin, log, and id combine their child node inputs by addition before operating.

Recall that every mathematical expression can be represented (non-uniquely) as a syntactic parse tree, where interior nodes represent mathematical operators and leaf nodes represent input values, which are usually variables or constants. To evaluate the represented expression, each operator node takes in the values of its children nodes as input. In

general, equation parse trees do not need to be perfectly balanced. However, by introducing an identity operator, as well as establishing a convention that unary operators will be applied on a pre-specified combination of two child nodes, every parse tree can be forced into a standard format that is both balanced and perfectly binary. An example of this process is shown in Figure 3.1, where two child nodes are combined using addition when fed to a unary operator parent node.

If the structure of an unfilled, balanced, binary parse tree is taken as a template, then it is possible to produce a mathematical function by identifying what operations and input quantities to "fill in" at each node of the tree. This is the essence of our SFL algorithm. The goal is to learn what symbols to place in each node of the symbolic parse tree that represents the expression that minimizes the given cost function. It is possible to approach this task in a way that is based on a combinatorial search, rather than deep learning [49]. However, in this work, we will show a method that is fully differentiable, and hence is receptive to deep learning techniques, such as training by back-propagation.

3.1.2 The Operator Function

We can define our method more rigourously as follows. Similar to [92], let U be a list of allowable unary operators $[u_1, \ldots, u_r]$ that map $\mathbb{R}$ to $\mathbb{R}$, and let V be a list of binary operators $[v_{r+1}, \ldots, v_k]$ that map $\mathbb{R}^2$ to $\mathbb{R}$, for a total of k operators that we will allow as symbols in our mathematical expressions. We define the "operate" function $\boldsymbol{p} : \mathbb{R}^2 \to \mathbb{R}^k$ by

$$\boldsymbol{p}(x_1, x_2) = [u_1(x_1 + x_2), \ldots, u_r(x_1 + x_2), v_{r+1}(x_1, x_2), \ldots, v_k(x_1, x_2)].$$

For example, if $U = [id, \sin, \exp\,]$ and $V = [\times]$, then

$$\boldsymbol{p}(x_1, x_2) = [x_1 + x_2, \sin(x_1 + x_2), e^{x_1+x_2}, x_1 x_2].$$

Note that we use id to refer to the unary identity function that returns its input value unchanged. In fact, in our implementation, it is equivalent to the addition operator on its two inputs, because we have defined the convention that unary operators are applied on two child node inputs by combining them into one by using their sum.

By using the $\boldsymbol{p}$ function, we are applying all of our possible operators to the two input values given to $\boldsymbol{p}$. We can use $\boldsymbol{p}$ to model the decision that takes place at every interior node in a symbolic parse tree template. Each interior node of a symbolic parse tree represents a single operator. The $\boldsymbol{p}$ function lets every operator have a chance of being selected. Now,

we would like to filter out all but one of these operators, so that only one is selected as the effective output value for this operator node. This can be done using a trainable gating operation.

3.1.3 Trainable Gating Operation

The gate can be set up as follows. Let $\boldsymbol{\omega}$ be a learnable weight vector in $\mathbb{R}^k$. The softmax function, $\boldsymbol{s}$, is

$$\boldsymbol{s}(\boldsymbol{\omega}) = \left[\frac{e^{\boldsymbol{\omega}_i}}{\sum_{j=1}^{k} e^{\boldsymbol{\omega}_j}} \right]_{i=1}^{k}.$$

Now $\boldsymbol{s}(\boldsymbol{\omega})$ is a nonnegative vector in $\mathbb{R}^k$ with entries summing to 1. The dot product $\boldsymbol{p}(h_1, h_2) \cdot \boldsymbol{s}(\boldsymbol{\omega})$ is then a convex linear combination of the outputs over all operators allowed by $\boldsymbol{p}$, skewed to give most weight to the operator at the index corresponding to the largest entry of $\boldsymbol{\omega}$. The choice of operator that is represented by a given node can thus be learned by updating the learnable weight vector $\boldsymbol{\omega}$ during training.

Although the softmax gate places most weight on a single entry of the vector of outputs passing through, it still retains nonzero contributions from all other operators in the output of $\boldsymbol{p}$. This can be corrected by adjusting the output of $\boldsymbol{s}(\boldsymbol{\omega})$ to become $\boldsymbol{s}'(\boldsymbol{\omega})$, where $\boldsymbol{s}'$ is a discretized form of softmax that returns a vector with 1 at the entry corresponding to the largest value of $\boldsymbol{\omega}$ and 0 at all other entries. One way to compute the discretized softmax is

$$\boldsymbol{s}'(\boldsymbol{\omega}) := \left[H_1 \left(\frac{\boldsymbol{s}(\boldsymbol{\omega})_i}{\max \boldsymbol{s}(\boldsymbol{\omega})} \right) \right]_{i=1}^{k}$$

so that

$$\boldsymbol{s}'(\boldsymbol{\omega})_i = \begin{cases} 1, & i = \arg\max(\boldsymbol{\omega}) \\ 0 & \text{otherwise} \end{cases}$$

where H_1 is a narrow hump function centred at 1, such as $H_1(x) = e^{-1000(x-1)^2}$. Since the division operator, maximum function, and hump function are all differentiable almost everywhere, the discretized softmax preserves the differentiability of our model that allows deep learning using gradient-based training methods.

In our SFL implementation, we use the standard softmax function for the first portion of training (around one quarter of the total iterations), and then shift to the discretized softmax for the remainder of the iterations. This allows the model to have some flexibility to navigate around the search space before settling down in a particular direction.

The framework we have described up until now is for a single operator node, which represents the foundational unit for the larger parse tree structure. This structure is illustrated in Figure 3.2, and is shown in simplified form in the top half of Figure 3.3. The bottom half of Figure 3.3 shows how the operator node structure fits into the broader architecture, which we will explain next.

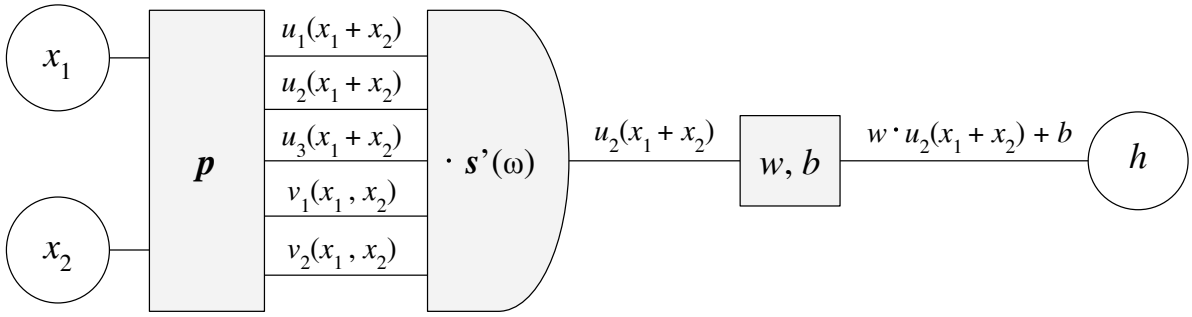

Figure 3.2: The architecture for a single operator node, the foundational unit for the larger parse structure in our SFL. In this example node, there are three unary operator options and two binary operator options, and the unary operator u_2 was selected by the discretized softmax gate. The way this node fits into the larger SFL model is shown in Figure 3.3.

3.1.4 The SFL Architecture

Let m represent the number of layers in the entire parse tree. The number of layers is a measure of how complex the represented mathematical expression is allowed to be. Trees with more layers form a richer space of mathematical functions, but provide a challenge by expanding the search space exponentially. For our SFL method, the number of layers must be fixed before constructing and training the model. If the number of layers is not known, the method can be repeated for several different choices for this number, and the resulting symbolic formulas can be inspected for the one that provides the best fit.

For $i = 0, \ldots, 2^m - 1$, define

$$h_i^{(0)} = \boldsymbol{w}_i^{(0)} \cdot \boldsymbol{x} + b_i^{(0)} \in \mathbb{R}$$

where $\boldsymbol{x} = [x_1, \ldots, x_d]$ is a vector of the variables in the mathematical expression being constructed, and $\boldsymbol{w}_i^{(0)} \in \mathbb{R}^d$ and $b_i^{(0)} \in \mathbb{R}$ are learnable parameters. This represents the

lowest layer of the tree, consisting of leaf nodes that denote raw numerical quantities before any symbolic operator has been applied on them. Each of these quantities is in the form of a learnable linear combination of all possible variables.

Working up the layers of the tree, as $n = 1, \ldots, m$, the value of each node is recursively defined as

$$h_i^{(n)} = w_i^{(n)} \left(\boldsymbol{s}' \left(\boldsymbol{\omega}_i^{(n)} \right) \cdot \boldsymbol{p} \left(h_{2i}^{(n-1)}, h_{2i+1}^{(n-1)}) \right) \right) + b_i^{(n)} \in \mathbb{R}$$

for each $i = 0, \ldots, 2^{m-n-1}$. Here, each $\boldsymbol{\omega}_i^{(n)} \in \mathbb{R}^k$ and $w_i^{(n)}, b_i^{(n)} \in \mathbb{R}$ are learnable weights whose values will be learned during the training process. Note that, unlike the first layer, where $\boldsymbol{w}_i^{(0)}$ was a d-dimensional vector, we now have $w_i^{(n)}$ as a real-valued weight for $n > 0$. This is because for the first layer, we do not take a discrete softmax (in order to allow unrestricted combinations of the input variables $x_1, \ldots, x_d$ to pass forward), so $\boldsymbol{w}_i^{(0)}$ takes the place of the vector $\boldsymbol{s}' \left(\boldsymbol{\omega}_i^{(n)} \right)$ that appears in subsequent layers.

The value at the root node of the tree, $h_0^{(m)}$, is the value of $\hat{f}(\boldsymbol{x})$, which is output of the mathematical expression represented by the tree, $\hat{f}$, when evaluated at the input vector $\boldsymbol{x}$.

To train the entire system, the function $\hat{f}$ can be used in a cost function $L(\hat{f})$ to determine the cost or error of the model. This error can be minimized by optimizing all of the training weights in the SFL using a gradient-based deep learning training method.

For the symbolic regression problem, we can set the cost function to be the expected mean squared difference between the true and predicted output values:

$$L(\hat{f}) = \mathbb{E} \left[\left(\hat{f}(\boldsymbol{x}) - y \right)^2 \right] \approx \sum_i \left(\hat{f}(\boldsymbol{x}_i) - y_i \right)^2 \tag{3.1}$$

In our SFL model, although each interior node of the symbolic parse tree is subject to a restrictive gate that allows only one operator choice to pass through, there is no such filter applied to the choice of variables in the leaf node layer. We found that in practice, it is often helpful to allow combinations of variables to pass as input into the first layer of operator nodes. However, to prevent the learned equation from growing too complex, we can apply an L_1 regularization term to the cost function to encourage sparsity [100, 101]. The weight for the regularization penalty in the cost function starts at 0, and grows larger toward the end of the training procedure, following a schedule similar to the one used in [69]. This way, the initial stages of training focus on exploration and the final stages focus on pruning and refinement.

At the end of the training procedure, which can be set to from 5000 to 20000 training steps or more, depending on the number of layers, variables, and operator choices, the SFL

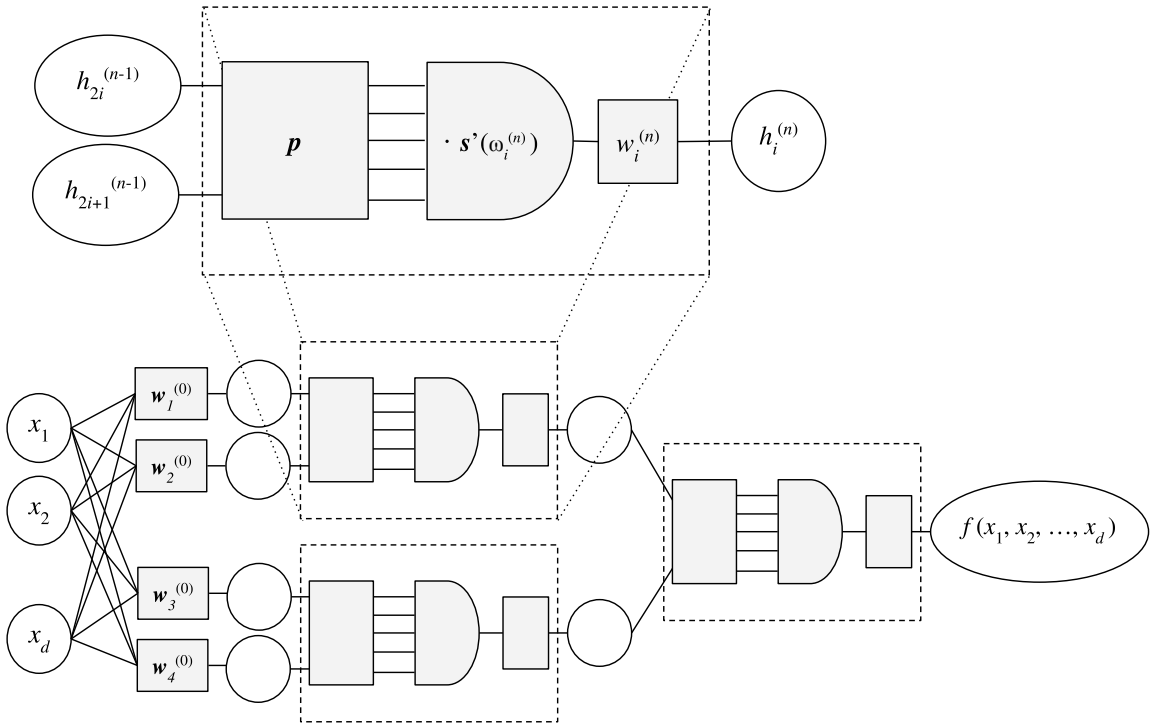

Figure 3.3: The architecture of our symbolic function learner (SFL). **Top:** The neural representation of a single operator (interior) node in the symbolic parse tree, replicated from Figure 3.2. The operator function $\boldsymbol{p}$ takes in two child nodes as input and applies all available operators on their values. The discretized softmax function $\boldsymbol{s}'$ is a gate that allows exactly one of these operators to pass through, determined by learnable weight $\boldsymbol{\omega}$. The output of this gate is then scaled by learnable weight w. (Note: bias scalars b are omitted from the diagram to save space.) **Bottom:** After an initial layer of leaf nodes which combine in a fully connected fashion, a series of operator nodes form the template of a balanced binary parse tree. The weight parameters determine how to interpret this tree as a single, well-formed symbolic mathematical expression f over multiple variables.

returns the symbolic expression represented by the final state of the parse tree. In other words, it interprets each interior tree node as the operator determined by the weight in its ω vector, and applies an affine transformation to each node defined by the corresponding w and b values. If the resulting function does not fit the training data exactly, it will usually be a close approximation. In practice, we repeat the training procedure 20 times with different random weight initializations, and take the result that attains the lowest mean square error on a left-aside validation set, which is usually randomly selected as about 30% of the training set.

3.2 Strengths and Advantages

As shown in Chapter 2, there are numerous methods for symbolic regression and symbolic function learning in general. Here are some of the advantages that our proposed SFL has to offer.

Configurability

Our SFL is highly configurable, easily and naturally allowing its users to select many of the hyperparameters that govern the equation search space. In addition to selecting the complexity of the sought equations by picking the number of layers in the equation parse tree, the user can determine which and how many operators to include as possible options at each internal node. These operators are not limited to ordinary arithmetic functions, either; the model is able to handle custom-defined operators as well, which can allow learning equations over different mathematical domains. For example, the SFL can learn vector equations on multidimensional input using vector operations, such as norm and dot product, and boolean equatons over binary datasets, using operators such as AND and OR. This level of configurability is not as conveniently possible for symbolic function learners that require pre-training langauge models in advance [15, 57].

Adaptability

Like other symbolic function learners, the SFL that we present here adapts very easily to many applications. In this thesis, we show how our SFL can be used to not only perform symbolic regression, but also learn differential equations, solve differential equations, perform integration, discover geometric properties, and more. One reason why our SFL is so

flexible is that, like EQL [69], we do not incorporate any heuristics or background assumptions into the architecture of our model; rather than forcing our method to specialize in certain domains such as physics or natural sciences [106, 105], it remains adaptable to a variety of applications and downstream tasks.

Model interpretability

The goal of symbolic regression is to arrive at a concise and interpretable model that describes a dataset, which is accomplished by producing a formula in the language of symbolic mathematics. Like all symbolic function learners, this is the objective that our method strives to achieve. However, our SFL has the added advantage that the model itself is highly interpretable. Every learnable weight in the SFL network plays a role directly in the discovered equation, allowing the model itself to be easily understood at all points throughout training. This white-box model enables a heightened sense of insight on how the function generated by the SFL is changing throughout the training procedure.

Learning of constants

One of the biggest challenges of learning symbolic expressions is the task of choosing the values of constants that appear within the expression. Unlike picking operators to form the skeleton of the equation, determining the choices of constants is not a discrete-valued decision task. Many symbolic regressors, including genetic models, face difficulty in reaching the right values for each constant that appears in the expression, sometimes requiring constant values to be directly provided [106] or determined as a separate optimization step [15]. Our SFL leverages the power of deep learning to update constant values as a step directly within the training procedure, allowing our method to work well on learning expressions with many real-valued, non-integer constants.

Ability to approximate

Symbolic function learners seek to solve an optimization problem over the space of symbolic functions. In many cases, a truly optimal symbolic function may be difficult or impossible to find. Our SFL, by means of its search strategy based on deep learning, always returns a symbolic function, even if it is not optimal; generally, the returned function will be a good approximation to the true function in terms of behaviour, at least over the domain of the training set. This is valuable because even when the true function is unavailable,

having a symbolic expression that defines a similar pattern can still offer some insight on the mechanics of the situation.

3.3 Limitations

Symbolic function learning is an inherently difficult problem, and like all other approaches, our SFL faces some limitations that impact its use.

Scalability

One of the biggest challenges faced by our SFL is scalability. Based on our investigations, the SFL faces difficulty in learning concise and useful functions if the number of tree layers, the number of variables, or the set of allowable operators grow too large. In these cases, the learning procedure takes too long and is less likely to converge to the correct function. Fortunately, many useful applications of symbolic regression do not require immensely complex formulas. Therefore, in practice, our method can be applied to a number of useful scenarios.

Convergence to local optima

The space of symbolic functions with real-valued constants is highly non-smooth and non-convex. While one symbolic expression may achieve a globally optimal minimum for a cost function in a given situation, there may be numerous other, entirely different, symbolic expressions that achieve locally minimal costs while located far away from the global minimum in the search space. As a result, every symbolic function learner faces the risk of abandoning the true optimal solution in favour of one less effective. We try to address this by incorporating several random restarts throughout our training protocol, but this is not a guarantee that the globally optimal function can or will be found.

Difficulty with some operators

As a model undergoing a gradient-based training method, our SFL is sensitive to operators that are not continuous or return extremely large values. In particular, operators such as $\div$, log, and exp have presented challenges to our SFL when they have been included in the set of allowable operators. This is a challenge faced by other neural network methods

as well [77], including EQL [69]. In this thesis, we focus mainly on applications that do not involve these challenging operators, although we will use them successfully for some examples shown in Chapter 4.

3.4 Evaluation

We conducted a variety of experiments to demonstrate the capabilities of our proposed SFL. The first experiment is a method comparison task that shows our SFL competing with representatives of two other major classes of regression algorithms on a large collection of artificial regression problems. The second experiment is a demonstration of the SFL performing on a custom task with real physical applications: learning the form of differential equations that admit a given solution function. These experiments are described in detail in the following sections.

Our experiments were performed using our own TensorFlow 2.2.0 implementation on Python 3.6, run on Microsoft Windows Server 2016 Standard OS with four Intel (R) Xeon (R) Gold 6126 CPU @ 2.60 GHz, 2594 Mhz, 12 core processors.

3.4.1 Symbolic Regression Comparison Task

To test our SFL's performance at general symbolic regression, we ran a large-scale comparison test against two common symbolic regression algorithms: a genetic evolution algorithm, as implemented in the standard Python package GPLearn, and an ordinary feed-forward neural network (Multi-Layer Perceptron, or MLP), as implemented in Python's SciKit package as MLPRegressor [82].

For this test, we randomly generated 150 instances of the symbolic regression task. For each of these instances, we constructed a symbolic formula by randomly filling in the nodes of a k-layer symbolic parse tree with suitable values. Each interior node was filled in with a randomly selected operator from the set $\{+, \times, \sin, \sqrt{\cdot}\}$, and each leaf node was filled in with a randomly selected variable ranging from x_1 to x_d. In addition, around 80% of the nodes the symbolic parse tree were randomly multiplied by a random constant value between -10 and 10, and around 80% of the nodes were increased by a random bias value between -10 and 10.

For each of these randomly generated symbolic formulas, we generated a dataset of 1000 random points with input values distributed uniformly at random from the region $[0, 5]^d$.

The output values were calculated by passing these points into the symbolic formulas. In this way, we constructed training datasets for a total of 150 symbolic regression instances, each with 1000 labelled training data points. In a similar way, we generated 1000 labelled test data points for each instance, where the test points were distributed uniformly from the expanded domain $[0, 10]^d$.

For each instance, we ran our SFL method, along with the GPLearn method (GPL) and the neural network method (MLP), to obtain three symbolic functions to fit the training dataset. We then computed the error score by evaluating the mean square difference between the true and predicted output values for each of the points in the test set.

We ran the entire procedure twice: once on symbolic expressions with parse trees having $k = 2$ layers and $d = 3$ variables, and once with $k = 3$ layers and $d = 2$ variables. In each case, we adjusted the SFL algorithm to use the number of variables and layers accordingly. The SFL was allowed to restart 20 times per equation, and the learned equation that performed best on a held-out validation set consisting of 300 randomly chosen points of the training data was returned as the resulting equation.

We used one hidden layer of 5 nodes for the MLP regressor in the first test, and two hidden layers of 5 nodes in the second test. The GPL regressor used all of the default settings from the GPLearn framework, although the set of allowable operators was restricted to the set $\{+, \times, \sin, \sqrt{\cdot}\}$.

In Tables 3.1 and 3.2, we show the proportion of the resulting symbolic formulas discovered by each method that obtained a mean square error less than specified cutoffs. Table 3.1 shows the results for the experiment on equations with 2 tree layers and three variables, and Table 3.2 shows the results for the experiment on equations with 3 tree layers and two variables. The complete plots showing the proportion of test equations that attained test errors less than every cutoff threshold are shown in Figure 3.4.

Method	Error <0.01	Error <0.1	Error <0.5	Error <1.0
GPL	3.3%	15.3%	36.7%	54.7%
MLP	2.7%	16.0%	38.7%	60.0%
SFL	**28.0%**	**41.3%**	**54.0%**	**67.3%**

Table 3.1: The proportion of equations with 2-layer trees over three variables which obtained a test error less than specified cutoffs.

From the tables, we see that for almost every error threshold of interest, our SFL was able to solve more symbolic regression instances than the competing methods. In

Method	Error <0.01	Error <0.1	Error <0.5	Error <1.0
GPL	3.3%	20.7%	40.7%	**61.3%**
MLP	2.7%	20.7%	42.7%	58.7%
SFL	**10.7%**	**28.7%**	**45.3%**	60.0%

Table 3.2: The proportion of equations with 3-layer trees over two variables which obtained a test error less than specified cutoffs.

particular, our method was much more successful at finding correct formulas for very low error thresholds than GPL or MLP.

3.4.2 Learning Differential Equations

In addition to finding symbolic functions to fit a given dataset of measurements, it is also possible to use an SFL to learn symbolic differential equations for which a given function is a solution. In this experiment, we show how our SFL is able to carry out this interesting task.

Suppose that we have access to a given function $y = f(x_1, \ldots, x_d)$, or, at least, are able to make measurements of y and its derivatives at any desired value of $(x_1, \ldots, x_d)$. Indeed, with oracle access to f, the values of its derivatives can be estimated at any point using approximations such as

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \boldsymbol{e}_i\varepsilon/2) - f(\boldsymbol{x} - \boldsymbol{e}_i\varepsilon/2)}{\varepsilon}$$

$$\frac{\partial^2 f}{\partial x_i^2} \approx \frac{f(\boldsymbol{x} + \boldsymbol{e}_i\varepsilon) - 2f(\boldsymbol{x}) + f(\boldsymbol{x} - \boldsymbol{e}_i\varepsilon)}{\varepsilon^2}$$

$$\frac{\partial^2 f}{\partial x_i x_j} \approx \frac{f(x + \varepsilon(\boldsymbol{e}_i + \boldsymbol{e}_j)) - f(x + \varepsilon(\boldsymbol{e}_i - \boldsymbol{e}_j)) - f(x + \varepsilon(\boldsymbol{e}_j - \boldsymbol{e}_i)) + f(x - \varepsilon(\boldsymbol{e}_i + \boldsymbol{e}_j))}{4\varepsilon^2}$$

where $\boldsymbol{e}_i$ is the unit vector in the ith direction, and ε is some small constant, such as 0.1.

We wish to discover a function g such that the differential equation

$$\frac{\partial^2 y}{\partial x_1^2} = g\left(x_1, \ldots, x_d, y, \frac{\partial y}{\partial x_1}, \ldots, \frac{\partial y}{\partial x_d}\right) \tag{3.2}$$

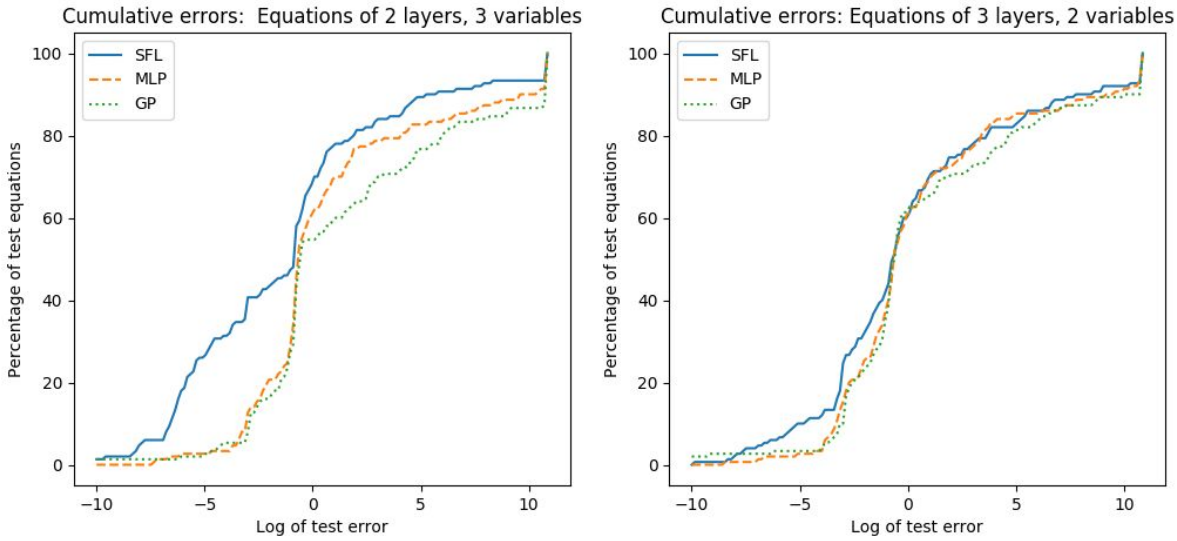

Figure 3.4: Cumulative error plots for each of the two comparative experiments, showing what proportion of each of the 150 test equations attained test errors less than each given value. Instances that failed grossly with test error larger than $e^{11} \approx 60,000$ had error scores truncated.

admits $y = f(x)$ as a solution. Note that, for simplicity, we are showing here a second-order differential equation in terms of first-order derivatives; the makeup and order of the derivatives can be generalized naturally as needed.

Let D be a set of randomly generated points over the region in $\mathbb{R}^d$ where the desired differential equation should hold. For each point $(x_1, \ldots, x_d)$ in D, compute or approximate the values of f, $\partial f/\partial x_i$ for $i = 1, \ldots, d$, and $\partial^2 f/\partial x_1^2$. Then construct the augmented dataset D^* where, for each point $(x_1, \ldots, x_d)$ in D, we have the input-output pair (u, v) in D^*, where

$$u = \left(x_1, \ldots, x_d, f(x_1, \ldots, x_d), \frac{\partial f}{\partial x_1}(x_1, \ldots, x_d), \ldots \frac{\partial f}{\partial x_d}(x_1, \ldots, x_d) \right) \in \mathbb{R}^{2d+1}$$

and

$$v = \frac{\partial^2 f}{\partial x_1^2}(x_1, \ldots, x_d) \in \mathbb{R}.$$

Now, the dataset D^* can be used in the cost function for an SFL which aims to find a

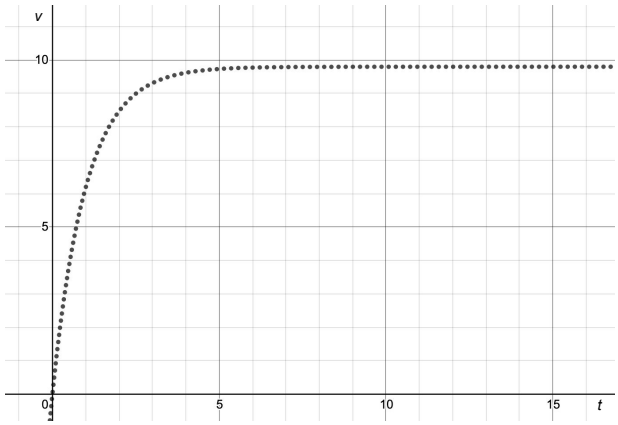

Figure 3.5: A plot of sample measurements of downwards velocity for an object falling while subject to air resistance.

symbolic expression g that minimizes

$$L(g) = \sum_{(u,v)\in D^*} (g(u) - v)^2 .$$

Note that this step uses the SFL in the same way as when performing symbolic regression. The equation learned by this algorithm, in the form $v = g(u)$, is a differential equation in the same format as Equation 3.2, as desired.

We now present some examples of using our equation-discovery system for learning differential equations if the desired solution is given. In the case of examples in physics, the equation describing an object's motion is often obtained from a differential equation. Here, we show two examples of the procedure in reverse. Note that the equations of motion can be obtained empirically by observed measurements in controlled experiments. This form of scientific discovery is not standard, but demonstrates the ability of our system to learn useful differential equations from known solutions.

Falling Object with Air Resistance

An object of mass m falling towards the earth while subject to non-neglibible air resistance will move downwards with velocity

$$v(t, m, k) = \frac{m}{k}\left(g - Ce^{\frac{-k}{m}t}\right)$$

where C and k are constants affected by the object's initial velocity and the strength of the air resistance. Suppose that this equation is known, as in a controlled environment, an object's speed can be measured at any time t. A plot of sample measurements for this phenomenon is shown in Figure 3.5.

Let $C = g = 9.80$. Then, if we make the variable substitution $b = k/m$, the velocity at time t is given by

$$v(t, b) = 9.80(1 - e^{-bt})/b.$$

We use this function v in our equation-discovery system to discover the first order differential equation

$$\frac{\partial v(t, b)}{\partial t} = -0.997b \cdot v(t, b) + 9.799$$

which can be understood as

$$\frac{\partial v(t, m, k)}{\partial t} = g - \frac{k}{m} v(t, m, k)$$

or simply

$$mv' = mg - kv,$$

which is known to be the application of Newton's second law for falling objects affected by the forces of gravity and air resistance.

1-D Damped Oscillator

A 1-dimensional damped oscillator of mass m moves along a trajectory given by

$$x(t, m, \alpha, k) = Ae^{\frac{-\alpha}{2m}t} \cos\left(\sqrt{\frac{k}{m} - \frac{\alpha^2}{4m^2}}t + \phi_0\right),$$

where α and k are constants describing the intensity of the dampening force and the springiness of the oscillator, and A and ϕ_0 represent the initial amplitude and phase shift. Suppose that this equation is known, as in a controlled environment, the position and velocity of the oscillator can be measured at any given time t. A plot of sample measurements for this phenomenon is shown in Figure 3.6.

Let $A = 1$ and $\phi_0 = 0$. For simplicity, we make the substitutions $b_1 = \alpha/m$ and $b_2 = k/m$. Then the path of the oscillator is given by

$$x(t, b_1, b_2) = e^{-b_1 t/2} \cos\left(t\sqrt{b_2 - \frac{b_1^2}{4}}\right).$$

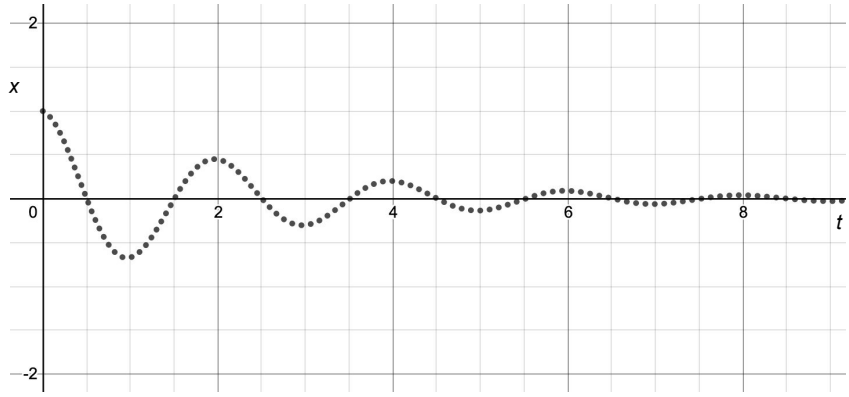

Figure 3.6: A plot of sample measurements of the position for an oscillating object subject to a damping force as time passes.

We use this function x in our equation-discovery system to discover the second-order differential equation

$$\frac{\partial^2 x}{\partial t^2} = -0.943 b_1 \frac{\partial x}{\partial t} - 0.972 b_2 x + 0.012$$

which, by reverse substituting b_1 and b_2, can be understood as

$$x'' = -\frac{\alpha}{m} x' - \frac{k}{m} x$$

or simply

$$ma = -\alpha v - kx,$$

where we use a and v to denote the acceleration x'' and velocity x' of the oscillator. Once again, we have recovered a known statement of Newton's second law for a specific situation.

Other Learned Differential Equations

To test the ability of our model at learning differential equations to fit functions without specific physics applications, we applied our method on a variety of different input functions. The resulting differential equations, along with error scores, are shown in Table 3.3.

Some of these results are what we might expect. The differential equation learned for Example #1, which indicates that $y = \sin(x)$ is a solution to $y'' = -0.985y - 0.333$, resembles the known differential equation for simple harmonic motion, $y'' = -y$. Other

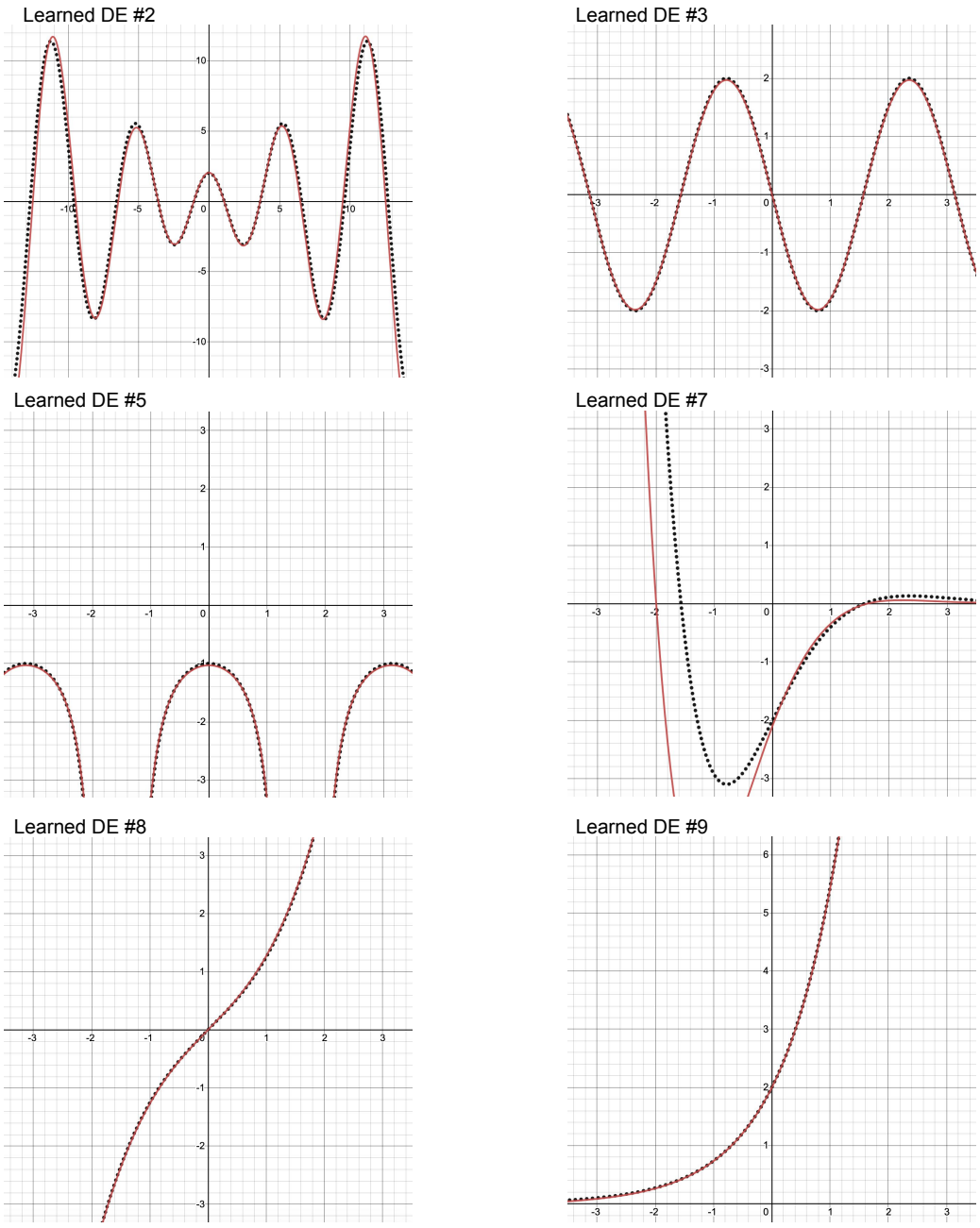

Figure 3.7: Some learned differential equations. Dotted lines show the derivatives of given functions, and solid lines are the learned differential equations evaluated at the given functions.

Table 3.3: Selected examples of learned differential equations that are satisfied by given functions.

#	$f(x)$	Learned differential equation for $f(x)$	Interval	Error
1	$\sin(x)$	$y'' = -0.985y - 0.333$	[-5, 5]	0.000104
2	$x\sin(x)$	$y'' = -2.088\sin(0.869x - 1.544)$ $-1.251y - 0.053$	[-5, 5]	0.298
3	$\sin(x)\cos(x)$	$y'' = -3.954y - 0.006$	[-5, 5]	0.00017
4	$2x^2 + 1$	$y' = 4.012\sqrt{\lvert 0.494 - 0.499y\rvert}$ $+0.012\sin(0.112y) - 0.039$	[-5, 5]	0.00099
5	$-\tan(x)$	$y' = -0.954y^2 - 1.029$	[-1, 1]	0.00113
6	$\frac{\cos(2x)}{40} - \frac{\sin(2x)}{20}$	$y'' = 13.70\sqrt{0.011 - 0.072y} - 1.678$	[-2, 2]	0.00213
7	$e^{-x}\sin(x)$	$y'' = 0.077x - 1.051y - 1.804y' - 0.293$	[-2, 2]	0.0516
8	$(x^2/4 + 1)^2$	$y' = 2.598x\sqrt{0.144y} + 0.037x$	[-3, 3]	0.0604
9	$2e^x + x + 2$	$y'' = 0.009y + 0.984y' - 0.991$	[-2, 2]	0.0083

results are more interesting: Example #4 shows that the parabola $y = 2x^2 + 1$ is a solution to an approximate differential equation that, when perturbed slightly, reveals

$$y' = 4\sqrt{|1 - y|/2} = \sqrt{8|1 - y|}.$$

We selected some examples to display visually in Figure 3.7. In these plots, we show how well the given functions fit the differential equations that were learned for them. The dotted line in each graph represents the required derivative of the given function, as it appears in the left side of the learned differential equation shown in Table 3.3. The solid line shows the (right side of the) learned differential equations from the table, when y is substituted with the true given function $f(x)$. We can see from the plots how well the given functions satisfy the differential equations that were generated for them.

3.5 Summary

In this chapter, we presented a novel architecture for neural-based symbolic function learning framework. This SFL, trained using deep learning, is configurable, adapts to address many tasks, is highly interpretable, enables learning of constant values, and produces approximate solutions in case it fails to find the exact ones.

We demonstrated how the SFL is useful for symbolic regression, performing competently in comparison with existing regression methods. We also presented the problem of learning differential equations as a novel application for symbolic function learners, and showed how our SFL produced useful results when used for this task.

In the following chapters, we will show how symbolic function learners, including our SFL, can be used for additional tasks in the pursuit of automated knowledge discovery.

Chapter 4

Symbolic Solutions to Differential Equations

Indeed We have created
everything in a measure.

Noble Qur'an, 54:49

Differential equations play a fundamental role in describing the physical laws that govern the natural world, and have been the subject of much study for centuries. More recently, artificial neural networks have gained popularity as they have been shown to demonstrate superior performance in a wide variety of tasks. Supported by the back-propagation techniques that make deep learning possible, there are few fields remaining in which neural networks have not made a contribution. Naturally, neural networks have been applied to solving differential equations, with promising results.

Many research studies have shown the ease with which neural networks can model solutions to previously unmanageable differential equations. In this case, the function satisfying the differential equation is not given in the form of an explicit mathematical expression. Rather, the solution takes the form of a highly complicated neural network model, making up for its lack of interpretability with its highly accurate fit.

Perhaps it is the success that neural networks have had with this approach to solving differential equations that has prevented progress in using deep learning to solve such problems in a different way: obtaining symbolic solutions in the form of readable, mathematical expressions.

A recent study on Deep Learning for Symbolic Mathematics [57] is one of the rare contributions that tries to address this gap. In that work, the authors demonstrate how using methods in deep learning can achieve state-of-the-art performance in solving differential equations and integration problems symbolically. Their method employs deep learning in an indirect way: rather than using deep learning to discover the solutions to differential equations themselves, their method involves constructing and making use of a language model that has been trained to *understand* the language of symbolic mathematics. Thus, the differential equations are not solved by using deep learning itself, but by means of a pre-trained model that had made use of deep learning as it was set up.

In contrast with this approach, we describe in this chapter a method that aims to use deep learning to solve problems in symbolic mathematics directly. We present a framework for solving differential equations that explores the space of candidate mathematical functions and uses deep learning techniques to identify the optimal solution for the problem. We show that our method also naturally extends to differential equations over multiple variables.

As part of this system, we use our symbolic regression algorithm as a neural-based Symbolic Function Learner (SFL) that is able to produce symbolic mathematical expressions that minimize a specified cost function. The key requirement of an SFL is that, given a class of symbolic functions F and a measure of cost on functions $L : F \to \mathbb{R}$, the SFL is able to discover

$$\arg\min_{f \in F} L(f). \tag{4.1}$$

Our implementation makes use of the SFL model presented in Chapter 3. This SFL seeks the minimizer of Equation (4.1) using a purely gradient-based search, allowing it to take advantage of all of the training techniques that neural networks have to offer.

As Lample and Charton admit [57], neural networks are rarely used to solve problems in symbolic mathematics because neural networks are poor at dealing with symbolic mathematics in general. We demonstrate that this weakness is not insurmountable. While our method may not compete with state-of-the-art algorithms for automated solving of highly complex differential equations, our proposed system has the advantage of being able to produce symbolic approximate solutions for every problem that it does not or cannot solve. These are symbolic expressions that may not necessarily solve the differential equations exactly, but whose values resemble those of the true solutions over a given interval.

Symbolic approximation functions are useful because they allow insight to the shape of a solution even when the solution itself cannot be found. Furthermore, they allow us to see which functions in the function space F are closest to the true solution when the true

solution itself does not lie in F. This is particularly useful in cases where the solution to a differential equation cannot be represented in terms of elementary functions.

In the following sections, we outline the core of our framework and enumerate some of the advantages of our system. We then show several experiments that demonstrate the power of our method and the utility of symbolic approximation functions. We are able to use our method to determine symbolic functions that approximately solve a number of members in the family of Lane-Emden equations [2], several of which have eluded closed-form solutions. We also show how our framework can be used to solve integration problems, and generate symbolic approximate integrals to a number of functions with non-analytic antiderivatives.

4.1 Method

A general form for representing nonlinear, ordinary differential equations is

$$g(x, y, y', y'', \ldots) = 0. \tag{4.2}$$

More generally, any partial differential equation (PDE) can be represented by the form

$$g\left(x_1, x_2, \ldots, x_d, y, \frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_d}, \frac{\partial^2 y}{\partial x_1^2}, \frac{\partial^2 y}{\partial x_1 \partial x_2}, \ldots\right) = 0. \tag{4.3}$$

The solution to this system is the mathematical function $f(x)$ that minimizes the expected error

$$L_A(f) = \mathbb{E}\left[\left|g\left(x_1, \ldots, x_d, f(x_1, \ldots, x_d), \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d}, \frac{\partial^2 f}{\partial x_1^2}, \frac{\partial^2 f}{\partial x_1 \partial x_2}, \ldots\right)\right|^2\right] \tag{4.4}$$

where x is distributed over the desired domain of f. Note that in most cases, it is sufficient to use discrete approximations for partial derivatives, such as

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \boldsymbol{e}_i\varepsilon/2) - f(\boldsymbol{x} - \boldsymbol{e}_i\varepsilon/2)}{\varepsilon}$$

$$\frac{\partial^2 f}{\partial x_i^2} \approx \frac{f(\boldsymbol{x} + \boldsymbol{e}_i\varepsilon) - 2f(\boldsymbol{x}) + f(\boldsymbol{x} - \boldsymbol{e}_i\varepsilon)}{\varepsilon^2}$$

$$\frac{\partial^2 f}{\partial x_i x_j} \approx \frac{f(x+\varepsilon(\boldsymbol{e}_i+\boldsymbol{e}_j)) - f(x+\varepsilon(\boldsymbol{e}_i-\boldsymbol{e}_j)) - f(x+\varepsilon(\boldsymbol{e}_j-\boldsymbol{e}_i)) + f(x-\varepsilon(\boldsymbol{e}_i+\boldsymbol{e}_j))}{4\varepsilon^2}$$

where $\boldsymbol{e}_i$ is the unit vector in the ith direction, and ε is some small constant, such as 0.1.

In some cases, the differential equation may be constrained by initial values for the function or any of its derivatives. These may be expressed in terms of points, $\{(x_i, y_i^{(n_i)})\}$, that must satisfy the solution to the differential equation. Frequently, these initial values represent boundary conditions that restrict the choices for the solution to the differential equation. In the present framework, we do not impose any requirement on whether these constraining points lie on the boundary of the domain or otherwise.

These constraints may be attained by a function $f(x)$ that minimizes the error

$$L_B(f) = \sum_i \left| f^{(n_i)}(x_i) - y_i^{(n_i)} \right|^2. \tag{4.5}$$

The solution to the differential equation can be therefore found by finding the function f that minimizes the error

$$Err = L_A(f) + \lambda L_B(f). \tag{4.6}$$

If f belongs to a class of functions that can be learned using methods based on gradient descent, then the differential equation can be solved by standard deep learning techniques, such as back-propagation. This is the way neural networks are typically used for solving differential equations.

However, if f is assumed to be in the form of a symbolic mathematical expression, then to use Equation (4.6) to learn the function via deep learning techniques, there must be a method for generating symbolic expressions that minimize a given cost function. This is the task that we aim to address using our symbolic function learner, described in Chapter 3.

4.2 Strengths and Advantages

Our proposed equation solving framework possesses the following strengths:

Ability to Approximate

Most automated systems for solving differential equations or performing integration are unable to produce a symbolic expression when given a problem that is outside of their

capacity to solve [75]. In particular, many differential equations and integration problems are known to have no closed-form analytic solution, which would be guaranteed to fail in these systems [52].

Our framework is able to provide a symbolic expression as output in every single instance. Since the SFL is performing an optimization search based on reducing an objective value, it does not distinguish between problems that are solvable or not, and always returns the expression found during the search process that attains the best objective score.

A powerful advantage of this fact is that our system can be used to generate optimal approximations for functions under user-imposed restrictions on expression complexity. Furthermore, our system excels at obtaining symbolic approximations for functions that cannot be expressed in terms of elementary functions at all, as will be shown in the Experiments section.

Modularity

The framework is modular and can be easily adapted as models for generating symbolic functions improve. Although we present one model for an SFL in this work, the framework can easily be applied using any other method for obtaining symbolic mathematical expressions that optimize a given cost function.

In particular, many symbolic regression algorithms are designed to find the symbolic expression that minimizes the mean square error (or some other fitness metric) over a given dataset. In some of these approaches, the regressor is performing an optimization problem over the search space of symbolic functions. This is the case for some genetic evolution based strategies, including Eureqa [93], as well as the methods that involve neural networks, including [92] and [106]. Our framework provides an alternative application where these same models can be used, with little adjustment, for a different task.

No Dependency on Language Model

The state-of-the-art method for using deep learning to solve problems in symbolic mathematics is given by Lample and Charton [57]. Their technique (which we call "LC"), while capable of producing outstanding results, necessarily requires learning a language model for symbolic expressions before it can be used.

Our model does not depend on having access to a trained language model. By dropping this requirement, we make a trade-off in performance: there are many differential equations that LC can solve exactly but our method will not.

On the other hand, the advantages of this trade-off are numerous. We do not need to train a hefty language model over millions of symbolic expressions, nor do we need to exert effort to obtain this trove of training data (in the form of labelled, input-output pairs of solved integrals and differential equations). The lack of dependency provides our compact model with the flexibility to adapt to new environments without training a new language model. For example, if we wish to change our set of allowable operations, we can do so at no extra cost and begin the SFL as normal.

Versatility

Our framework was designed with differential equations in mind, but can be easily applied to a variety of other tasks with no extra effort. In addition to first and second order diffential equations, the following problems can be tackled using our framework by simply using the appropriate choices for the function $g(x, y, y', y'')$ of Equation (4.3):

- **Integration:** Compute the antiderivative of $p(x)$ by setting $g(x, y, y', y'') = y' - p(x)$.
- **Functional equations:** Solve the functional equation $h(x, y) = 0$ by setting the objective "differential" equation as $g(x, y, y', y'') = h(x, y)$.
- **Compute inverse functions:** Obtain an expression to approximate the inverse of the function $p(x)$ by setting $g(x, y, y', y'') = x - p(y)$.
- **Compute roots:** Obtain a approximate numerical root of the function $p(x)$ by setting $g(x, y, y', y'') = p(y)$.
- **Symbolic regression:** Obtain a symbolic expression to fit a dataset of points $\{(x, y)\}$ by defining $g(x, y, y', y'')$ as 0 (declaring it as an irrelevant aspect during training) and providing the dataset of points for fitting as the set of initial value conditions, guiding the optimization process.

4.3 Limitations

The main limitation in our model is the same that affects all approaches used for symbolic function learning: the space of all possible mathematical functions is so vast that the task of identifying the one expression best suited for a given objective function is very likely to be an NP-hard problem [106]. This difficulty is further compounded by the fact that neural

networks are not naturally suited to approaching problems in symbolic mathematics [57]. Although our SFL discovers accurate or useful functions in many cases, it is susceptible to getting trapped in local minima and returns suboptimal results on several occasions. This problem grows more severe as the number of tree layers or allowable operations increases.

4.4 Evaluation

To test our system, we performed a number of example tasks on a variety of problems. In each task, we run our system when given a function $g(x, y, y', y'')$ that encodes the problem we are trying to solve. Unless stated otherwise, the SFL algorithm is run using the unary operators $U = [id, \sin, \sqrt{\cdot}]$ and the binary operator $V = [\times]$. In order to avoid illegal argument errors, we automatically compute the absolute value before entering any value as input to operations defined only on the positive half-plane, such as the square root function. Note that adding an explicit addition operator is not necessary by the way we defined id in the operator function $\boldsymbol{p}$ of our SFL.

We run our method 20 times on each task, each time using 5000 randomly generated points within the domain of the desired function f to train our model. Each run proceeds for 6000 iterations and returns the function represented by the parse tree at the final iteration. We observed that more iterations than this were not needed whenever the SFL was on track to reaching convergence. During the first 1250 (= 25%) of iterations of each run, we use the standard form of softmax function $\boldsymbol{s}$, and switch to the discrete form $\boldsymbol{s}'$ for the remainder of the training. This allows the model to more freely explore the function space during early stages of training, before settling down towards a single expression structure and fine-tuning constants for the remaining iterations.

The output of our system is the function produced by the SFL that obtained the lowest validation error out of all 20 runs, as per Equation (4.6), on a held-out validation set of 1500 randomly chosen points from the training data points.

We use a TensorFlow 2.2.0 implementation on Python 3.6, run on Microsoft Windows Server 2016 Standard OS with four Intel (R) Xeon (R) Gold 6126 CPU @ 2.60 GHz, 2594 Mhz, 12 core processors.

4.4.1 Lane-Emden Differential Equations

The Lane-Emden equations are a family of equations relating to hydrostatic equilibrium that have applications in astrophysics [2]. The general form of the homogeneous Lane-

m	True Expr.	Learned Expression	Error on $[1, 5]$	Error on $[0.1, 10]$
0	$1 - x^2/6$	$1.000 - 0.166x^2$	0.000064	0.00016
1	$\sin(x)/x$	$0.91 - 0.85\sin(0.04x^2 + 0.01x$ $+0.67\sin(0.18x - 0.03) - 0.04)$	0.327	5.320
2	None	$0.485\sin\left(\frac{2.0x+0.5}{0.3x+1.8} + \frac{0.176x+0.007}{0.142x+0.006} + 0.094\right)$ $+0.507$	0.0047	0.157
3	None	$0.980 - 0.407\sin(0.170x + 0.016)$ $-0.001x^2 - 0.069x$	0.0935	0.271
4	None	$-0.053 + \frac{2.95\sqrt{0.7087x+0.4733}+2.79}{0.77x^2+1.50x+4.50}$	0.00044	0.103
5	$\frac{1}{\sqrt{1+\frac{x^2}{3}}}$	$1.00 - (1.01\sin(0.18x - 0.01) + 0.025)$ $\div \left(\frac{-1.45x-0.6}{0.1-1.6x} + 0.08\right)$	0.00989	0.0815

Table 4.1: The Lane Emden equation, shown in Equation (4.7), has known solutions for $m = 0, 1$, and 5, but not other values of m. Our system is able to produce symbolic expressions to approximate solutions for all values of m.

Emden differential equation of order m is

$$y'' + \frac{2}{x}y' + y^m = 0 \tag{4.7}$$

where $y(0) = 1$ and $y'(0) = 0$ are initial conditions.

This family of differential equations is interesting, among other reasons, because closed-form solution to this equation are known for $m = 0, 1$, and 5, but not for other values of m. It is not known if analytic solutions exist in these cases. Instead, researchers resort to computational approximations for these equations [80].

We ran our system on the Lane-Emden equation for values of m ranging from 0 to 5 by setting $g(x, y, y', y'')$ as given in Equation (4.7). For $m = 1$ onwards, we added the division operator $\div$ to our set of allowable binary operations. Our method obtained the symbolic expressions shown in Table 4.1, graphed visually in the right panel of Figure 4.1. The error score over the interval $[a, b]$ is a measure of how close the values of the g function are to 0 on the interval when using the learned expression f, computed using the integral $\int_a^b |g(x, f(x), f'(x), f''(x))|^2 \, dx$.

The Lane-Emden equations are meant to be applied over the semi-infinite domain $[0, \infty)$, but for computational purposes we restricted the domain of training to be within $[0.1, 4]$. Points around $x = 0$ were excluded in order to avoid the singularity in Equation (4.7). Despite the fact that the points given in initial value conditions fall outside

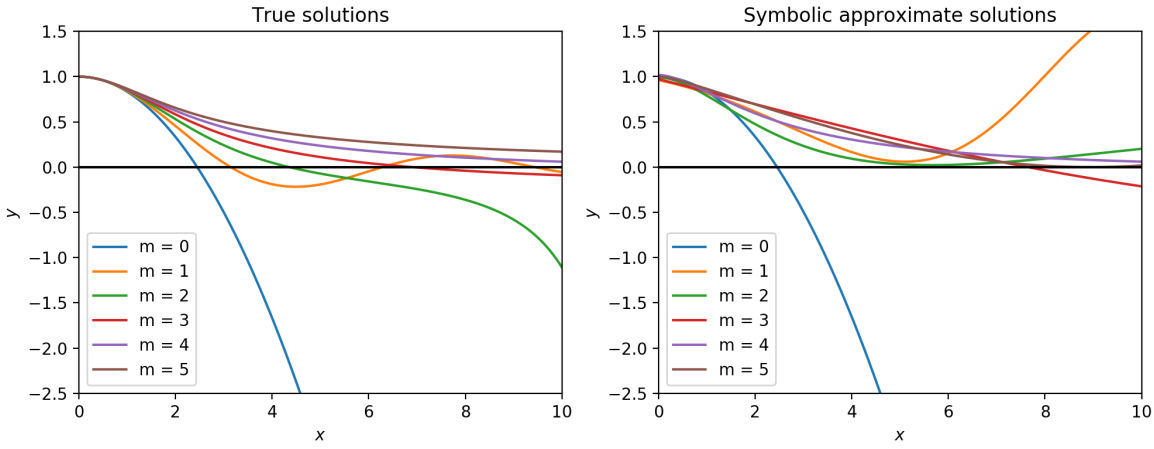

Figure 4.1: Graphs of Lane-Emden equations, comparing the true solutions (left) with the symbolic approximations obtained using our model (right). Note that the symbolic approximation functions are able to replicate the general behaviour of the true functions, while constrained to using only the prespecified operations in mathematical expressions of limited complexity.

the domain of the training points, our system is still able to effectively abide by the given constraints.

As seen in Table 4.1 and Figure 4.1, our system was able to produce competent symbolic approximations for the Lane-Emden equations even when no symbolic solution is known to exist. Aside from when $m = 1$, the approximation quality was very good over the interval $[1, 5]$.

4.4.2 1-Dimensional Wave Equation

We also tested our system on a number of mutivariate partial differential equation problems and demonstrate the results.

The motion of wave travelling in one spatial dimension over time can be modelled by a function $u(x, t)$ that satisfies the PDE

$$\frac{\partial^2 u}{\partial t^2}(x,t) = c^2 \frac{\partial^2 u}{\partial x^2}(x,t) \tag{4.8}$$

for $x \in [0, \pi]$, $t > 0$. This type of motion can be found in particles vibrating around a rest position along a single direction [96].

Consider the case where $c = 1$ and we are given the boundary conditions

$$u(0, t) = 0$$
$$u(\pi, t) = 0$$

for $t > 0$, and initial conditions

$$\begin{aligned} u(x, 0) &= 0 \\ \frac{\partial u}{\partial t}(x, 0) &= \sin x \end{aligned}$$

for $x \in [0, \pi]$.

This system has the exact solution $u(x, t) = \sin x \sin t$. Our method produced the symbolic solution

$$\hat{u}(x, t) = 1.0002 \sin(1.000x) \sin(0.9998t).$$

This result is represented visually in Figure 4.2. In this figure, as well as the ones following it, the left-most graph is a plot of the learned symbolic function. The central graph shows the residual error from the learned function, that is, the value of g evaluated at $\boldsymbol{x}$ as in $L_1(f)$ used in Equation 4.4. The right graph demonstrates how the learned function fits the specified boundary conditions. The plots show how the residual errors are small and the boundary conditions are satisfied well for this example.

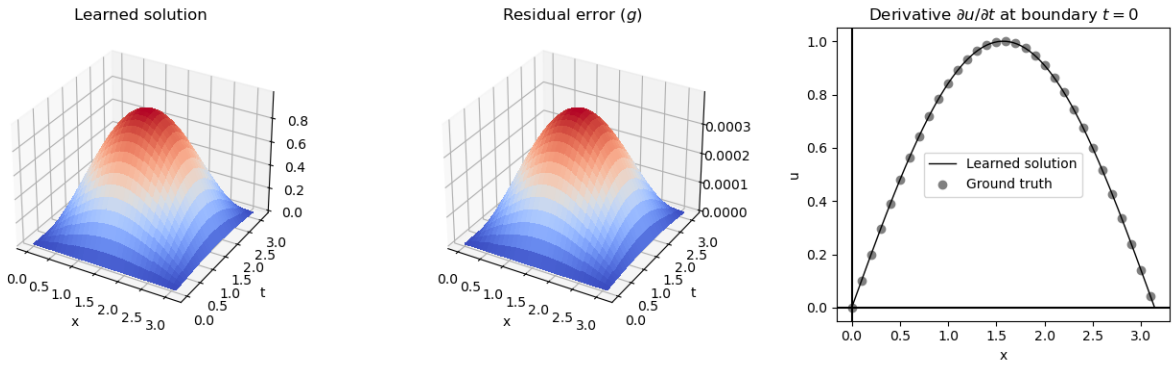

Figure 4.2: The learned solution to the wave equation (4.8), with residual error and boundary values.

4.4.3 1-Dimensional Heat Equation

The diffusion of heat through a medium along a single spatial dimension over time can be modelled by a function $u(x,t)$ that satisfies the PDE

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{4.9}$$

for $x \in [0,\pi]$, $t > 0$. For example, this equation might be used to model how an uneven initial distribution of heat disperses and levels out along a rod of uniform composition.

Consider the case given by the boundary conditions

$$u(0,t) = 0$$
$$u(\pi,t) = 0$$

for $t > 0$, and initial conditions

$$u(x,0) = \sin x$$

for $x \in [0,\pi]$.

This system has the exact solution $u(x,t) = e^{-t}\sin x$. Our method produced the symbolic solution

$$\hat{u}(x,t) = (1.005e^{-0.994t} - 0.005)\sin(0.9996x + 0.001t).$$

This result is represented visually in Figure 4.3. Once again, the residual errors are small and the boundary conditions are met.

4.4.4 Fokker-Planck Equations

The Fokker-Planck equation is a general model describing the behaviour of particles undergoing Brownian motion, with applications ranging from astrophysics to economics [89]. The general format of a linear Fokker-Planck (FP) equation, also called a forward Kolmogorov equation, is

$$\frac{\partial}{\partial t}u(x,t) = -\frac{\partial}{\partial x}A(x,t)u(x,t) + \frac{\partial^2}{\partial x^2}B(x,t)u(x,t) \tag{4.10}$$

with initial condition $u(x,0) = f(x)$, where $A(x,t)$ and $B(x,t)$ (known as drift and diffusion coefficients, respectively), along with $f(x)$, are specified real-valued functions.

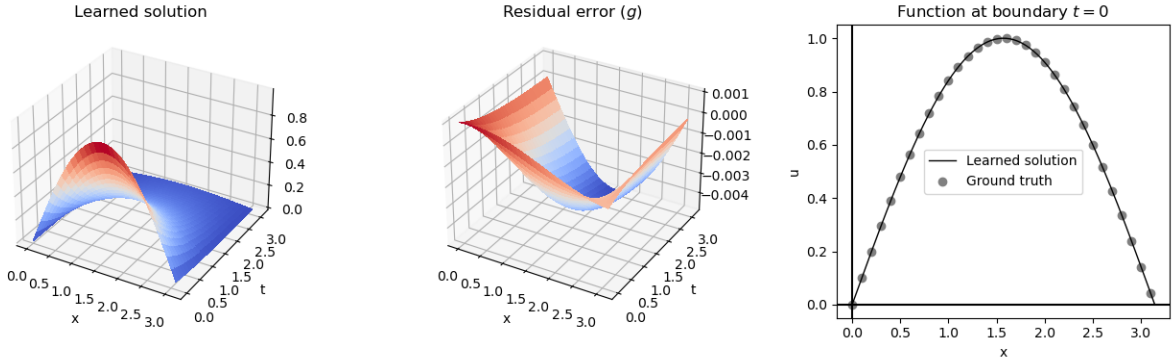

Figure 4.3: The symbolic solution to the heat equation (4.9), with residual error and boundary values.

In order to fit the structure of Equation (4.3), we can reformulate Equation (4.10) as

$$g\left(u, \frac{\partial u}{\partial t}, \frac{\partial x}{\partial t}, \frac{\partial^2 x}{\partial t^2}\right) = \frac{\partial u}{\partial t} - \left(\left(\frac{\partial^2 B}{\partial x^2} - \frac{\partial A}{\partial x}\right) u + \left(2\frac{\partial B}{\partial x} - A\right)\frac{\partial u}{\partial x} + B\frac{\partial^2 u}{\partial x^2}\right) = 0. \quad (4.11)$$

An alternative version for an FP equation, known as the backward Kolmogorov equation, is

$$\frac{\partial}{\partial t}u(x,t) = -A(x,t)\frac{\partial}{\partial x}u(x,t) + B(x,t)\frac{\partial^2}{\partial x^2}u(x,t) \quad (4.12)$$

with initial condition $u(x,0) = f(x)$, where $A(x,t)$, $B(x,t)$, and $f(x)$ are specified real-valued functions. Note that this formulation is already close to matching the format of Equation (4.3), and does not need to be heavily rearranged.

We will look at three common examples of FP equations with known analytic solutions. These examples have been investigated in several existing works on FP equations, including those by [56], [31], and [46].

Example 1

Consider the case of Equation (4.10) over $x, t \in [0,1]$ where $A(x,t) = -1$ and $B(x,t) = 1$, subject to the initial condition $f(x) = x$.

This system has the exact solution $u(x,t) = x + t$. Our method produced the symbolic solution

$$\hat{u}(x,t) = 1.0001x + 1.0001t.$$

This result is represented visually in Figure 4.4.

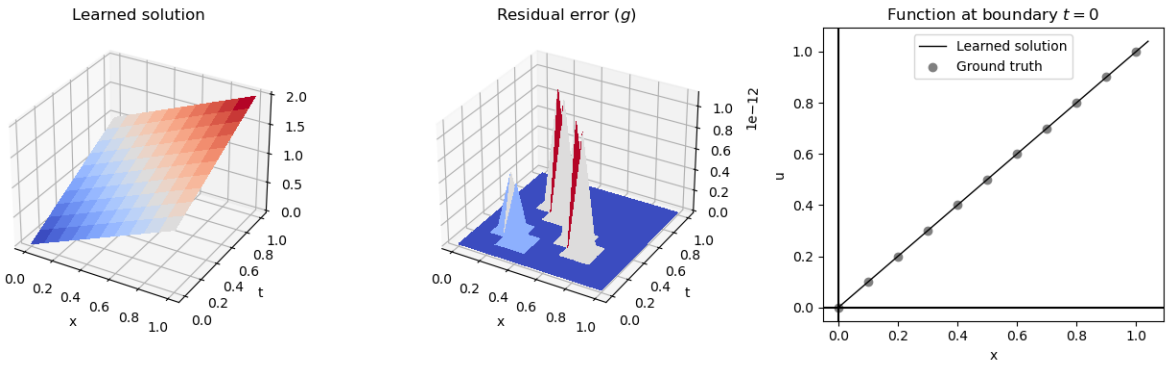

Figure 4.4: The symbolic solution to FP Example 1, with residual error and boundary values.

Example 2

Consider the case of Equation (4.10) over $x, t \in [0, 1]$ where $A(x, t) = x$ and $B(x, t) = x^2/2$, subject to the initial condition $f(x) = x$.

This system has the exact solution $u(x, t) = xe^t$. Our method produced the symbolic solution

$$\hat{u}(x, t) = (0.972x - 0.001t + 0.002)e^{1.024t} + 0.029x - 0.002.$$

This result is represented visually in Figure 4.5.

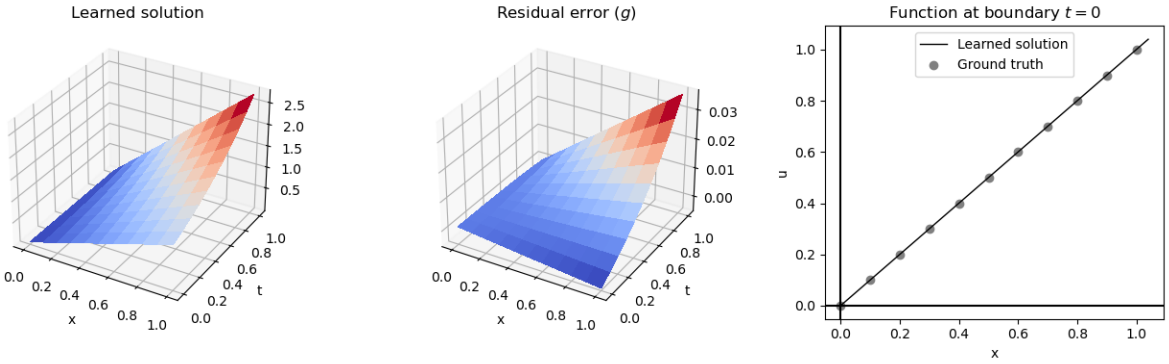

Figure 4.5: The symbolic solution to FP Example 2, with residual error and boundary values.

Example 3

Consider the case of Equation (4.12) over $x, t \in [0, 1]$ where $A(x, t) = -(x + 1)$ and $B(x, t) = x^2 e^t$, subject to the initial condition $f(x) = x + 1$.

This system has the exact solution $u(x, t) = (x + 1)e^t$. Our method produced the symbolic solution

$$\hat{u}(x, t) = (0.923x - 0.006t + 0.866)e^{0.011x+1.121t} + 0.057x + 0.136.$$

This result is represented visually in Figure 4.6.

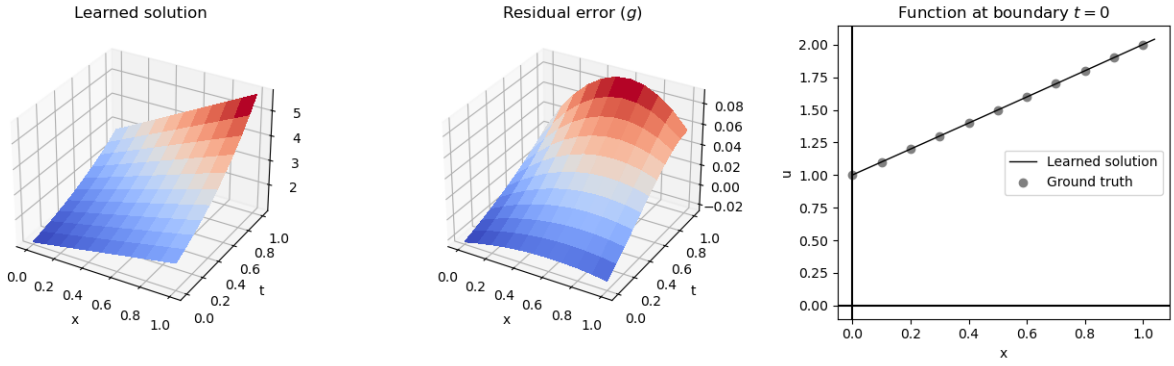

Figure 4.6: The symbolic solution to FP Example 3, with residual error and boundary values.

4.4.5 Integration for Non-Elementary Functions

Many existing solvers compute the antiderivatives of analytically integrable functions. However, our method has the property of being able to generate symbolic approximations for antiderivatives for all functions $f(x)$, including those that do not have integrals that can be expressed in terms of elementary functions, by choosing our objective differential equation as $g(x, y, y') = y' - f(x)$.

We demonstrate this ability of our method on four common examples of functions that do not have analytic antiderivatives. The results shown in Table 4.2 use error function $Err = \frac{1}{b-a} \int_a^b \left(\int_0^x f(t)dt - \hat{F}(x) \right)^2 dx$, where $\hat{F}$ is our approximation for the antiderivative of f over $[a, b]$.

Name	Integrand, f	Learned antiderivative $\hat{F}$ of f	Interval I	Error on I
Bell curve	e^{-x^2}	See $\hat{B}(x)$ in text	[-2, 2]	0.00185
Elliptic integral	$\sqrt{1-x^4}$	$1.50\sin\Big(0.15x^2+0.58x$ $+1.28\sqrt{\lvert 0.053x+0.05\rvert}+0.05\Big)$	[-1, 1]	0.00094
Nested sin	$\sin(\sin(x))$	$\sin(0.532x)(0.241x+\sin(0.541x))$	$[-\pi,\pi]$	0.00234
Root of sin	$\sqrt{\sin(x)}$	$0.938x\sin(\sqrt{0.494x})$	$[0,\ \pi]$	0.0103

Table 4.2: The symbolic expressions found by our method for integrals that cannot be expressed in terms of elementary functions.

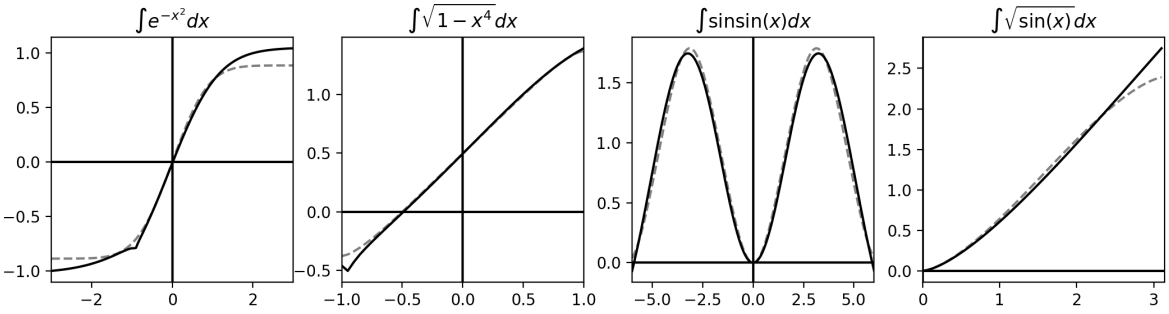

Figure 4.7: Each of the above graphs compares the true antiderivative (dotted line) with the symbolic approximation produced by our method (solid line) for a function whose integral is not expressible in terms of elementary functions. Constants of integration were manually selected to line up the curves vertically for easy comparison. The symbolic expressions we found are shown in Table 4.2.

The Bell Curve

It is well known that there is no elementary function whose derivative is e^{-x^2}. We applied our method to approximate this integral $B(x) = \int_0^x e^{-t^2} dt$ over the set of operators $[+, \times, \sqrt{\cdot}, \sin]$ on the interval $[-2, 2]$ using a 3 layer SFL parse tree.

The resulting expression obtained was

$$\begin{aligned}\hat{B}(x) \;=\; & 1.039\sin(0.969|0.982\sin(0.045x-0.105)\\ & -1.083\sin(0.120x+0.102)-0.107|^{0.5}\\ & +1.300\sin(1.224\sqrt{|0.046x+0.042|}\\ & +1.097\sin(0.451x-0.039)-0.206)-0.577)\\ & +0.016\end{aligned}$$

This is certainly a complex expression, but it fits reasonably well over the interval $[-2, 2]$, as can be seen in Figure 4.7.

Although the definition of $\hat{B}(x)$ is cumbersome, it offers insight that would not have been visible by inspection. Using this function as a starting point, we visually detected and manually applied perturbations to obtain the expression

$$\begin{aligned}\widehat{erf}(x) = 0.545\sin(&\sqrt{\sin(0.1368x+0.0883)+0.2120}\\ &+1.300\sin(\sin(0.5162x+0.1931)-0.5716)).\end{aligned}$$

While by no means elegant, this expression is simpler than the previous one, and also possesses a useful property: on the interval $[-1, 3]$, it is almost identically equal to the function $erf(x) = \int_{-\infty}^{x} e^{-t^2/2}/\sqrt{2\pi}dt$, representing the area under the normal distribution curve. The corresponding error score over this interval is

$$\int_{-1}^{3}\left(\widehat{erf}(x) - \int_{-\infty}^{x}\frac{e^{-t^2/2}}{\sqrt{2\pi}}dt\right)^2 dx = 0.000006464.$$

Thus, in the spirit of [115], the semi-automatically generated symbolic function $\widehat{erf}(x)$ is almost an exact match for the cumulative distribution function for the standard Gaussian distribution, as seen in Figure 4.8, and hence provides an analytic alternative to lookup tables when computing probabilities of normally distributed events.

Elliptic Integral

Another function whose integral is not expressible in terms of elementary functions is the elliptic integral $y = \int_0^x \sqrt{1-x^4}dt$.

We tested our framework on this integral and include the results in Table 4.2 and Figure 4.7.

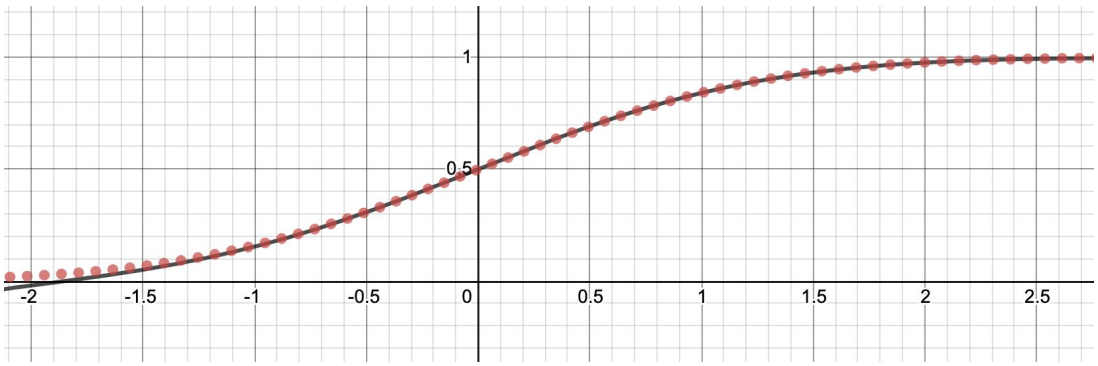

Figure 4.8: The true CDF for the Gaussian distribution (dotted line) is approximated well by the analytic function $\widehat{erf}(x)$ discovered by our method (solid line).

Trigonmetric Integrals

Two more functions with integrals that cannot be expressed in elementary form are $y = \sin(\sin(x))$ and $y = \sqrt{\sin(x)}$.

We applied our framework on both of these functions and and include the results in Table 4.2 and Figure 4.7.

4.4.6 Computing Functional Inverse

As one final example, we use our system to produce a symbolic approximation to the inverse of $c(x) = x^3 + x + 1$ in terms of the sin function. Using a 2-layer SFL trained over the interval [-2, 2], we obtain the approximate inverse function

$$c^{-1}(x) \approx 0.86 \sin(0.524x + 0.383 \sin(0.872x - 0.86) - 0.076) - 0.321$$

which we illustrate in Figure 4.9.

This example demonstrates how our framework can be used not only for differential equations, but also to find solutions to functional equations and compute inverse functions.

4.5 Summary

We have presented a framework for using deep learning techniques to obtain symbolic solutions to differential equations. Based on a number of experiments, we have shown that

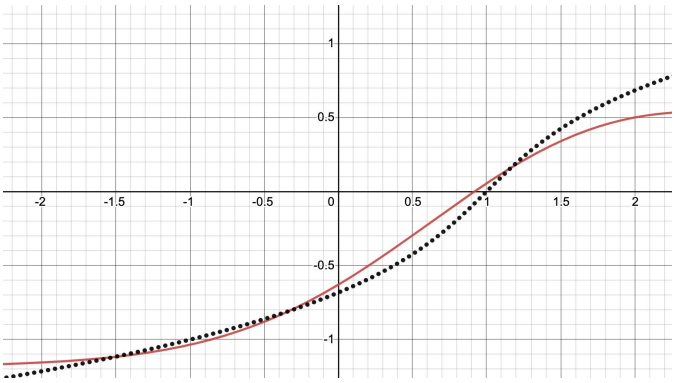

Figure 4.9: This graph compares the true inverse (dotted line) with the symbolic approximation produced by our method (solid line) for the cubic function $c(x) = x^3 + x + 1$.

our method is able to effectively supply symbolic function approximations to problems that have no elementary symbolic solutions.

Our system is open to accepting symbolic function learners other than the SFL we presented in Chapter 3. It would be good to see how competitive symbolic regression tools would perform within our framework. Solving differential equations (and other tasks in symbolic mathematics) would be a good way to test the abilities of symbolic regression algorithms, beyond the normal problems that they are usually tested against.

In the next chapter, we introduce another application for using symbolic functions to learn properties of the natural world, but now with an approach based on unsupervised learning.

Chapter 5

Unsupervised Learning of Mathematical Relationships

You did not recite any book before it, nor did you write it with your right hand...

Noble Qur'an, 29:48

Symbolic regression is not the only way to discover symbolic mathematical functions that describe relationships between variables in numerical datasets. Although it allows for much more flexibility than more constrained methods of regression, symbolic regression is still tied to the framework of representing a specified output variable as a function of given input variables.

In this chapter, we explore the possibility of using a symbolic function learner to discover symbolic relationships between variables in a way that removes some of the structural requirements of symbolic regression. In doing so, we can use the proposed method for the following objectives:

1. Discover symbolic expressions that describe relationships between variables when none of the variables is explicitly identified as the "output".

2. Learn mathematical properties that characterize a given set of objects using only "unlabelled" measurements or observations.

3. Perform the standard task of symbolic regression, but now allowing the learned equation to be implicitly defined in terms of the output variable.

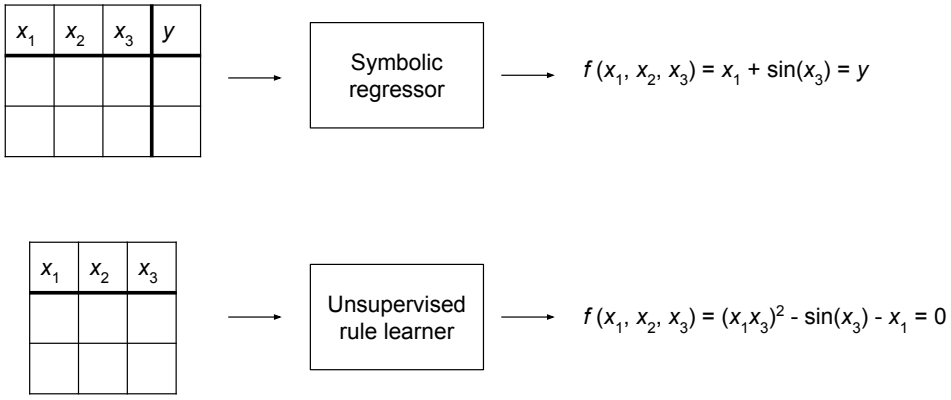

Figure 5.1: A comparison of the methods for symbolic regression and unsupervised rule learning, with example outputs. **Top:** Symbolic regression takes in labelled input-output data and finds a formula to predict the output variable as a function of the input. **Bottom:** Unsupervised rule learning takes in unlabelled data and finds an equation equalling 0 that is satisfied by all of the datapoints.

The method for achieving these objectives, as well as illustrative examples and experiments, will be presented in the sections that follow.

5.1 Method

We present our method in two parts. The first part is the general approach for unsupervised rule learning, which we later demonstrate by rediscovering several identities in geometry. The second part is a specific application of the approach that performs symbolic regression in a way that the dependent variable is defined implicitly.

5.1.1 Unsupervised Symbolic Equation Discovery

We now present a method by which a symbolic function learner may be used for equation discovery beyond the normal framework of symbolic regression. In particular, we show how to use a symbolic function learner to discover mathematical relationships between different variables without using a specific regression structure, which would require input and output variables to be explicitly specified in the training data. By doing so, we are able to learn symbolic expressions that show implicit relationships between variables in a given dataset, where no single variable is treated as an explicit "output" value. This is a way of generating a list of "properties" satisfied by elements in a given set of data, using an unsupervised, discovery-based approach. Effectively, we are learning a mathematical way to characterize the members of the set D.

Intuitively, our method works by constructing an augmented dataset that combines the original, "real" data with new, "fake" data, and learns a symbolic function to act as a classifier to discriminate between the two. If the fake data is generated such that it does not share common properties with the objects described by the real data, then the discriminating function will take the form of a symbolic expression that defines a property of the real data. In this way, we have discovered a symbolic relationship between the variables in the real dataset.

This process will be explained more rigourously in the following sections.

Real data and fake data

Concretely, suppose we are given a dataset D of n points over a set of real-valued variables, $x_1, \ldots, x_d$. We wish to find some mathematical expression $f : \mathbb{R}^d \to \mathbb{R}$ that defines a relationship between these variables over the dataset, such that $f(x_1, \ldots, x_d) = 0$ for every point $(x_1, \ldots, x_d) \in D$. In other words, we would like to discover some underlying structure in D by finding any mathematical rules that the points in D tend to obey.

Of course, there are trivial functions that would certainly satisfy the condition as specified, such as the zero function that returns 0 at every point in $\mathbb{R}^d$, as well as any symbolic function that identically simplifies to zero, such as $f(x_1, \ldots, x_d) = x_1/x_1 + x_2 - x_2 - 1$. These functions do not reveal any properties exclusive to the points in D. We are interested in finding functions f that equate to 0 over the dataset D, but are not identically equal to 0; in particular, f should generally not return 0 at points that are not in D. Thus, our goal is to find $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$f(x_1, \ldots, x_d) \begin{cases} = 0, & (x_1, \ldots, x_d) \in D \\ \neq 0, & (x_1, \ldots, x_d) \notin D. \end{cases} \tag{5.1}$$

Our approach is to obtain a new set of n' points from $\mathbb{R}^d$, called D', that contains only points that are not in D. In particular, the points in D' do not satisfy the properties that are special to the points in D. We can think of the set D' as containing "fake" data that our learned equation should distinguish from the "real" data contained in D.

In practice, if the points in D are believed to follow some relatively uncommon structure, the set D' can be generated at random. However, in general, our method seeks a function to discriminate between the points in D and D', so D' can be chosen more precisely whenever the situation calls for it.

The two sets D and D' are then merged together into a new, labelled set D^*, where the points originating from D have a label of 1 and the points from D' have a label of 0. Hence, D^* is a set of $n + n'$ points in $\mathbb{R}^{d+1}$, where each point is of the form $((x_1, \ldots, x_d), y)$ for some $y \in \{0, 1\}$.

Symbolic function learning

The set D^* is now a labelled, real-valued dataset, and as such, is suitable as input for a symbolic regression algorithm. By applying a symbolic function learner on D^*, we would seek a symbolic function $g : \mathbb{R}^d \to \mathbb{R}$ such that $g(x_1, \ldots, x_d) = 1$ for $(x_1, \ldots, x_d)$ in D and $g(x_1, \ldots, x_d) = 0$ for $(x_1, \ldots, x_d)$ in D'. Note that this is not (yet) the same as what we were originally seeking, because the labels are reverse from what we wanted in Equation (5.1).

To fix this label reversal, we make one slight modification when performing our symbolic regression: we insert a hump function, centred at 0, just after computing the output value of the symbolic function but before computing the cost function as in Equation (3.1). Thus, the symbolic regression is trained to minimize the difference between $H_0(f(x_1, \ldots, x_d))$ and y for each point $((x_1, \ldots, x_d), y)$ in D^*, where $H_0 : \mathbb{R} \to [0, 1]$ is a hump function such as $1/(1 + 10x^2)$. This smooth curve, which maps 0 to 1 and values far from 0 to values close to 0, is illustrated in Figure 5.2.

The purpose of the hump function is two-fold. First, it swaps the output of g from the labels used in D^* to match the desired output of f as in Equation (5.1), so that points in D map to 0 instead of 1. Second, it converts the real-valued output of f into a $[0, 1]$-valued output to match the range of labels in D^*. Importantly, the hump function is only close to 1 when its input is close to 0, and is close to 0 everywhere else. Hence the hump function

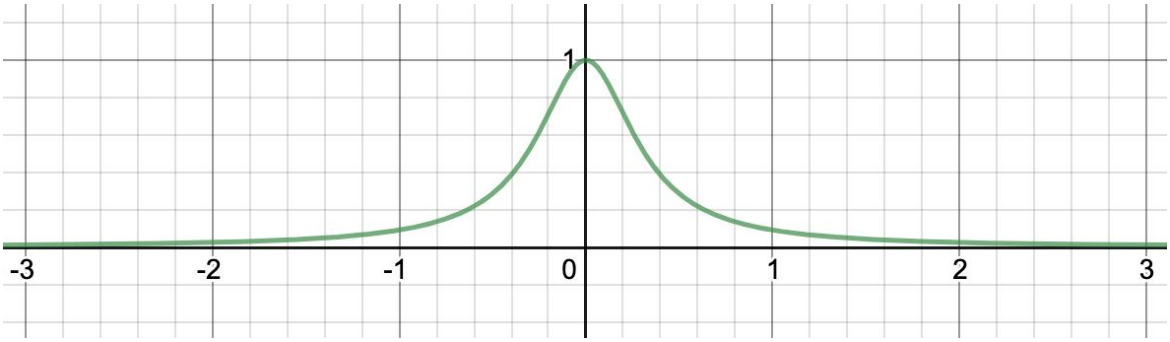

Figure 5.2: A sample hump function, $H_0(t) = 1/(1 + 10t^2)$.

flattens "all values other than 0" to go towards the single output value 0, which is the desired behaviour of f.

Thus, by using the hump function, this method will seek out functions f that return values close to 0 when the output label is 1 (which is when the input to f is in D), and values anywhere other than 0 when the output label is 0 (which is when the input to f is in D'). This is the condition that we desired the function f to satisfy in Equation (5.1).

By this method, we can discover symbolic functions that demonstrate properties of the set D in an unsupervised way. Every time the method is run with a new random initialization, a new equation will be learned. The fitness of the equation can be measured by computing the regression error of $H_0(f)$ on the a test dataset constructed in a similar manner as D^*. In practice, we can run this method hundreds of times and look at the discoveries generated by the equations with the lowest error scores, which should correspond with mathematical properties that hold for all points in D and no points outside of D.

5.1.2 Implicit Function Regression

Although the method described above can be used to learn relationships for an unlabelled dataset, it can also be used as an approach for the supervised learning task of symbolic regression. In fact, this method offers a way to perform symbolic regression to learn functions that are implicitly defined with respect to the target variable.

Suppose that X is a dataset of points of the form $\{(x_1, \ldots, x_d, y)\}$. To perform implicit symbolic regression on this dataset, apply the method by letting the points in the "real" dataset D be the points of X treated as elements of $\mathbb{R}^{d+1}$, and letting the "fake" dataset D' be points in $\mathbb{R}^{d+1}$ that are *not* on the desired curve, such as points chosen at random.

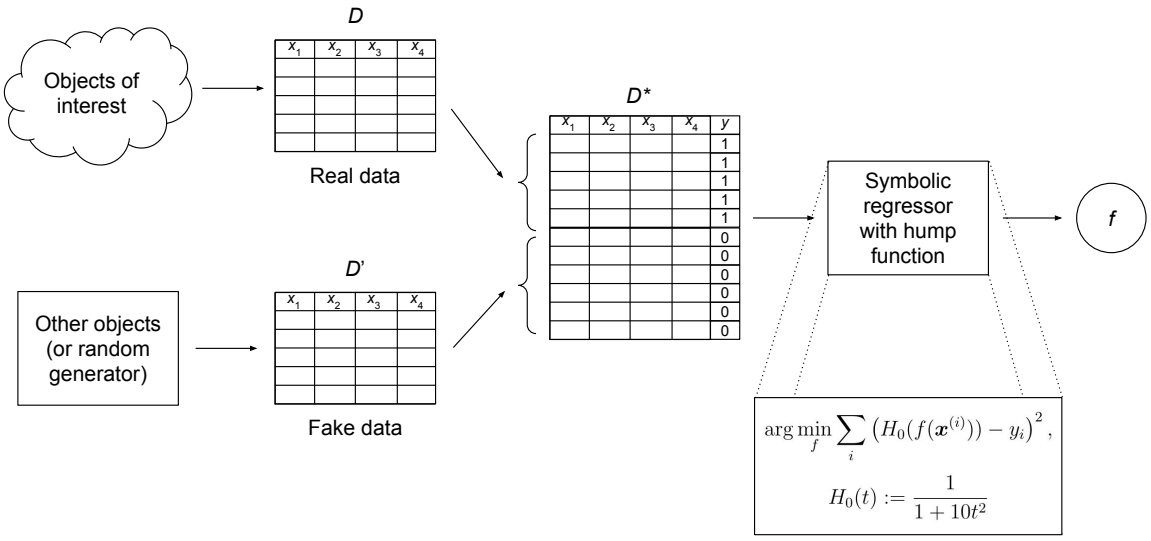

Figure 5.3: A summary of our unsupervised learning framework. A collection of real data is combined with a collection of fake data, with datapoints assigned output values of 1 or 0 indicating whether they are real or fake. This combined dataset is passed into a symbolic regressor equipped with a hump function. The resulting learned formula is the desired learned property.

In this way, the method will seek to find a symbolic formula f such that the expression $f(x_1, \ldots, x_n, y)$ evaluates to 0 for all points $(x_1, \ldots, x_n, y)$ in D. Thus, the expression obtained to fit the dataset X will be in terms of the output variable y implicitly rather than explicitly, allowing a kind of symbolic regression that is not normally possible for methods that only seek to find curve-fitting functions of the form $y = f(x_1, \ldots, x_d)$. We will demonstrate some examples of this later in this chapter.

5.2 Strengths and Advantages

This method learns symbolic mathematical expressions to fit datasets in a way that overcomes many of the restrictions of normal symbolic regression. Some of the advantages offered by our approach are the following.

Non-specification of variable dependency

The dataset does not need a specific variable designated as the output to be expressed in terms of all the other (input) variables. This removes the requirement on the researcher to make an assumption about a casual relationship between variates. Instead, no variable is explicitly labelled as independent or dependent, and relationships are sought that may interrelate some or all of the variables.

Purely data-driven discovery of mathematical properties

The method can be used to learn properties or structure behind classes of objects in a data-driven way. Instead of relying on insight or logic to discover invariants between variables in a dataset, this method generates defining mathematical properties in a purely automated procedure, once the measurements are taken and supplied. Rather than replacing a human-guided investigation, this method allows a supplementary search for properties that is less likely to face influence from human biases or errors.

Implicitly-defined symbolic regression

The symbolic regression application of our method naturally enables learning symbolic formulas defined implicitly in terms of the output variable, which is not often possible for many symbolic function learning algorithms directly.

5.3 Limitations

We have observed some areas where our method has faced difficulty in practice. Some of them are as follows.

Scalability

As with the tasks presented in previous chapters, scalability remains an issue. In particular, we face difficulty when applying our SFL to seek relationships over a large number of variables. In these situations, our model is more likely to fail to converge to equations with low test error. One way we can get around this, when the number of measured variables is large, is by randomly selecting a subset of variables to use at each attempt of rule learning.

While this allows our method to discover properties in high-dimensional datasets, it is unable to recover any formulas that contain many or all of the variables within them.

Performance on standard symbolic regression

Although our method is theoretically capable of performing standard symbolic regression, where the specified output variable is explicitly defined in terms of the independent variables, we observed that the method does not often produce the correct equation in practice. It seems that the learned equations tend towards functions that are defined implicitly, and face more difficulty in finding true equations than in the standard approach to symbolic regression. This may be due to a variety of reasons, ranging from a poor choice of hump function to a potential intrinsic weakness of binary classification approach towards regression. This is an issue that could benefit from further investigation.

5.4 Evaluation

We demonstrate the effectiveness of both the general unsupervised learning method and the implicit symbolic regression application method through a variety of experiments. In all cases, we use the symbolic function learner defined in Chapter 3 as a basis for our implementations.

We restarted the training procedure 500 times to obtain 500 learned equations for each task. These equations were sorted by their test error according to the SFL, and the representative examples with lowest error were selected for inclusion in the results that follow. When reporting learned equations, terms with coefficients of magnitude less than 0.05 were omitted to improve clarity.

5.4.1 Learning Geometric Relationships

To test our unsupervised symbolic expression learning method, we attempted to re-discover some of the famous laws of geometry from scratch.

Triangles

For our first experiment, we let D_T be a set of data encoding the side lengths and angles of 1000 different triangles. Our goal was to discover universal relationships between the

side lengths and angles within triangles in general.

To construct each datapoint of D_T, three random points in $\mathbb{R}^2$ from the region $[0, 5] \times [0, 5]$ were generated and treated as vertices of a triangle. We can refer to the sides lengths of this triangle as a, b, and c, with corresponding angles as A, B, and C. The datapoint was then constructed to consist of the 12-tuple

$$\begin{aligned}&(a, b, c, \sin(A), \sin(B), \sin(C), \cos(A),\\ &\cos(B), \cos(C), \tan(A), \tan(B), \tan(C)) \in \mathbb{R}^{12},\end{aligned}$$

with one such datapoint obtained for each random triangle generated. The counter dataset, D'_T, consisted of 1000 tuples in R^{12} containing entries chosen uniformly at random from the range [0, 5] for the first three entries, [-1, 1] for the next six entries, and [-5, 5] for the final three entries.

In this way, we trained our method to discover mathematical relationships between the side lengths and the trigonometric ratios of the angles in triangles. For this experiment, we allowed up to 3 layers in the SFL tree, and allowed equations to use the operators $\{id, \times\}$. In each training session, to reduce the complexity of the regression problem, we randomly selected a subset of 4 to 6 of the 12 variables to run our model on.

Three of the properties our method discovered are listed below:

$$-0.350a\sin(B) + 0.356b\sin(A) + 0.139\sin(A) - 0.094\sin(B) = 0 \tag{5.2}$$

$$0.420\sin^3(C) + 0.116\sin^2(C) + 0.356\sin(C) + 0.854\cos^2(C) - 0.930 = 0 \tag{5.3}$$

$$0.259\sin(C)\cos(C) + 0.535\sin(C)\tan(C) - 0.540\tan(C) = 0 \tag{5.4}$$

Note that the learned equation (5.2) simplifies to

$$\frac{\sin(A)}{a + 0.269} = 0.983\frac{\sin(B)}{b + 0.390},$$

an equation very closely resembling the law of sines, namely,

$$\frac{\sin(A)}{a} = \frac{\sin(B)}{b}.$$

This relationship is known to hold for all triangles, and would normally be deduced using geometric and algebraic reasoning. Our method was able to discover this law of geometry by observing several examples of randomly generated triangles, and optimizing a formula to fit the observations, with little to no human intervention.

Interestingly, not every result from this experiment corresponds to a true property for triangles. For example, our method identified the following property to hold on the triangles in D_T:

$$\begin{aligned} &0.133\sin(A) + 0.140\sin(B) + 0.125\cos(A)^2 \\ &+ 0.086\cos(A) + 0.144\cos(B)^2 + 0.091\cos(B) \\ &+ 0.110\cos(C) - 0.440 = 0 \end{aligned}$$

Although this expression is nearly accurate when the angles A, B, and C are taken from most acute triangles, it does not hold as an identity in general. We have a similar situation for the learned equations (5.3) and (5.4), which simplify to

$$\sin^3(C) + 0.276\sin^2(C) + 0.848\sin(C) = 2.214(1 - 0.918\cos^2(C))$$

and

$$\sin(C)\cos(C) = 2.2085\tan(C)(1 - 0.991\sin(C))$$

respectively. In these equations, the learned expressions appear to work very well by achieving low test error, but are not true for all values of the angle C. Since the equations (5.3) and (5.4) contain only one variable, we can graph the left sides of the learned equations as functions of C, as shown in Figure 5.4.

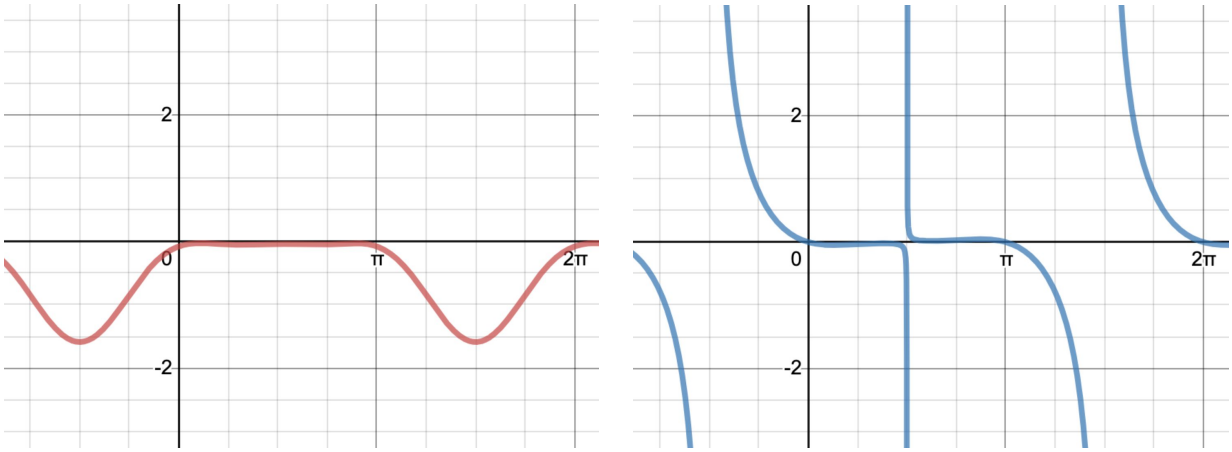

Figure 5.4: These plots show the values of the expressions shown in the left sides of the learned equations (5.3) and (5.4), left and right respectively, as the value of the angle C varies from $-\pi$ to 2π.

When presented visually, we see that the learned expressions are indeed very close to 0 for almost all values of C between 0 and π, but not necessarily anywhere else. Indeed, the

angles in a triangle cannot exceed π radians, so our unsupervised algorithm never looked at values of angles outside of this interval. Our method, when trained only on measurements from real triangles, was able to discover relationships between the trigonometric functions that are only valid for angles that come from real triangles.

Unrestricted Angles and Trigonometric Identities

In order to learn more general relationships between the trigonometric functions when the domain is not limited to angles between 0 and π, we constructed a new dataset, $D_{2\pi}$, consisting of points in $\mathbb{R}^3$ of the form

$$(\sin(x), \cos(x), \tan(x))$$

for 1000 values of x chosen uniformly at random from the interval $[0, 2\pi]$. The counter dataset, $D'_{2\pi}$, consisted of tuples in R^3 containing entries chosen uniformly at random from the range [-1, 1] for the first two entries and [-5, 5] for the final entry.

This new dataset allows for angles to vary over their entire domain, and would therefore allow our method to capture trigonometric identities that are valid for all real angles. For this experiment, we allowed up to 4 layers in the SFL tree, and once again used the allowable operators $\{id, \times\}$. Two of the learned equations include:

$$-2.633\sin^2(x) - 2.632\cos^2(x) + 2.640 = 0 \tag{5.5}$$

$$\begin{aligned} &-0.173\sin^2(x)\tan(x) - 0.796\sin^2(x) - 0.423\sin(x)\cos(x)\tan(x) \\ &-1.218\cos^2(x) + 0.076\sin(x)\cos(x) + 0.176\sin(x) \\ &-0.242\cos^2(x)\tan(x) - 0.168\cos(x)\tan(x) + 0.173\tan(x) + 1.221 = 0 \end{aligned} \tag{5.6}$$

It is interesting to note that the Pythagorean identity, namely, $\sin^2(x) + \cos^2(x) = 1$, was was never discovered when the dataset was only based on triangle measurements. However, once the domain expanded to include all possible angles, this famous equation was neatly identified, as seen in equation (5.5), which simplifies to

$$1.0004\sin^2(x) + \cos^2(x) = 1.003.$$

Equation (5.6) shows a typical low-error output from our method. This equation does not demonstrate a clean and simple relationship between the dataset variables. Instead, it resembles a large, complicated expression that must be carefully simplified before it can

reveal a mixture of various approximate identities that explain the low error value. In the case of Equation (5.6), the learned equation simplifies as

$$\begin{aligned} & 0.144(\sin(x) - 0.955\cos(x)\tan(x)) \\ & - 0.142\tan(x)(\sin^2(x) + 1.40\cos^2(x) - 1) \\ & = \Big((0.652\sin(x) + 0.346\cos(x)\tan(x))\sin(x) \\ & + 0.998\cos^2(x) - 1\Big) \end{aligned} \tag{5.7}$$

which resolves to several repeated approximations of the known trigonometric relationships $\sin^2(x) + \cos^2(x) = 1$ and $\tan(x)\cos(x) = \sin(x)$. The graph of Equation (5.6) is shown in Figure 5.5.

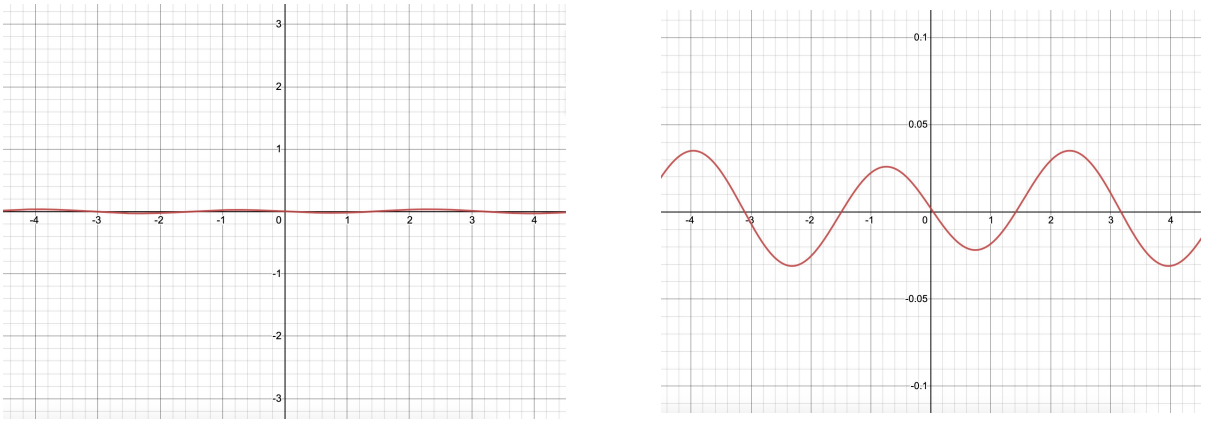

Figure 5.5: The graph of the left side of Equation (5.6) as a function of x. **Left:** The y-axis is scaled 1:1 with the x-axis. **Right:** The y-axis is stretched to the range [-0.1, 0.1], showing the minuscule fluctuations in the curve more clearly.

Right Triangles

As another experiment, we attempted to discover properties that apply specifically to right-angled triangles, in contrast with triangles that do not contain right angles.

In this framework, we constructed D_R by randomly sampled measurements for 1000 right angle triangles, obtaining points in $\mathbb{R}^{12}$ of the same form as in the Triangles experiment above. To guarantee that these triangles contain a right angle, we selected the side lengths

a and c of each triangle by sampling two values randomly from the interval $[0, 5]$, labelling them so that $c > a$, and then computing $b = \sqrt{c^2 - a^2}$. This ensures that each triple (a, b, c) corresponds to the side lengths of a triangle with side lengths of at most 5 and with angle $C = \pi/2$.

For D'_R, we used data for 1000 randomly generated triangles as in dataset D for the Triangles experiment above, noting that the probability for a triangle randomly generated this way to contain a right angle is sufficiently negligible.

We ran our method using the $[id, \times]$ operators on symbolic parse trees of 3 levels over 4 variables. Many of the discovered formulas quickly picked up on the fact that, since $C = \pi/2$ for every triangle in D_R, we have that $\sin C - 1 = \cos C = 0$. Two of the less trivial relationships we were able to obtain are the following:

$$0.205a^2 + 0.202b^2 - 0.215c^2 = 0 \tag{5.8}$$

$$-4.508\sin(B) + 4.481\cos(A) + 0.013 = 0 \tag{5.9}$$

which simplify, respectively, to

$$1.015a^2 + b^2 = 1.064c^2$$

and

$$\sin(B) = 0.994\cos(A) + 0.003.$$

Equation (5.8) is the well-known Pythagorean Theorem for right triangles, $a^2 + b^2 = c^2$. Equation (5.9) demonstrates the relationship between the sine and cosine of complementary angles: in right triangles, where $A + B = \pi/2$, it follows that $\sin(A) = \cos(B)$, or simply $\sin(x) = \cos(\pi/2 - x)$. Note that the dataset D_R would only have given enough information for us to conclude this relationship for angles x in the interval $(0, \pi/2)$, even though it holds in general for all $x \in \mathbb{R}$.

5.4.2 Implicit Regression Examples

This section evaluates our method's ability to perform symbolic regression. We present examples of equations of curves learned by our method that are not the result of functions, but are defined implicitly in terms of both x and y.

Equation of a Circle

The equation

$$x^2 + y^2 - 25 = 0$$

describes the graph of a circle of radius 5, centred at the origin.

To apply the proposed method, we let the real dataset D be the set of points of the form $(\cos(t), \sin(t))$ for 500 values of t uniformly sampled from $[0, \pi]$, and we let the fake dataset D' be a set of 500 randomly chosen points in $[-5, 5]^2$. We let the allowable operators for the SFL include $[id, \times, \sin]$, and instructed it to look for equations of 3 tree layers. Our method was able to produce the implicit equation

$$-0.088x^2 - 0.089y^2 + 2.217 = 0$$

which simplifies to

$$0.993x^2 + 1.000y^2 + 25.000 = 0,$$

as desired. The graph of the true equation and the learned curve are depicted on the left side of Figure 5.6.

It is interesting to look at the some of the other equations that were found during the discovery process, such as

$$0.1353 - 3.7516 \sin(0.0290x^2 + 0.0289y^2 - 0.6908) = 0,$$

or

$$\sin(0.0290x^2 + 0.0289y^2 - 0.6908) = 0.0361.$$

Note that although this curve, depicted in the right side of Figure 5.6, fits the desired points exactly within the training region $[-5, 5]^2$, it clearly includes many other points outside of the region. This example demonstrates one of the limitations of the proposed system for implicit symbolic regression, which is also a common limitation for all regression algorithms: there is no guarantee that the learned function will behave in a desired way far beyond the region of the training interval.

Oblique Ellipse

The equation

$$5x^2 - 6xy + 5y^2 - 12 = 0$$

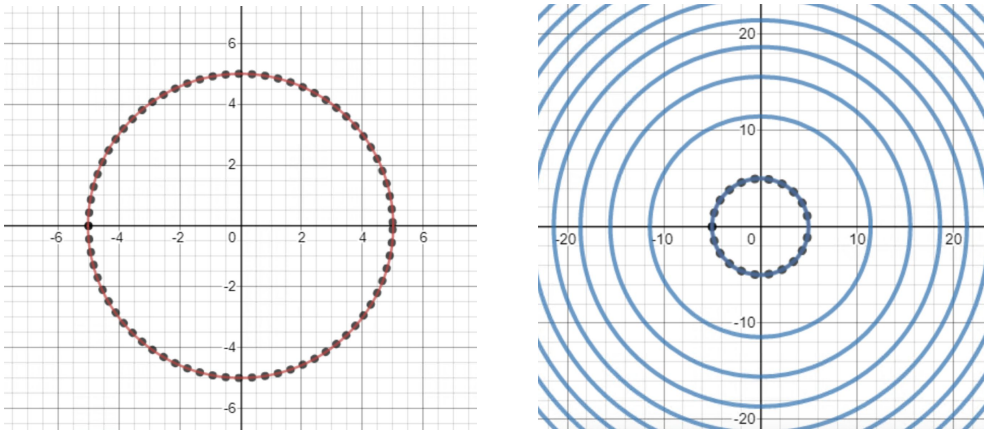

Figure 5.6: **Left:** the correctly learned implicit equation for the circle $x^2 + y^2 = 25$. The dotted points are the true values and the smooth curve is the learned equation. **Right:** a sample "incorrect" equation obtained using our method. Although the regression was accurate in learning the innermost circle, the resulting equation includes many larger circles as well.

describes the graph of an ellipse centred at the origin rotated by an angle of $\pi/4$, fully contained within the region $[-2, 2]^2 \subseteq \mathbb{R}^2$.

We applied our method to perform symbolic regression in order to try and recover this equation from a dataset of points along the curve. Our training dataset consisting of 100 randomly selected points along the ellipse for the real set D and 100 randomly selected points in the region $[-2, 2]^2$ as the false set D'. We allowed our SFL to use the operators $[id, \times, \sqrt{\cdot}]$ and looked for equations of at most 3 tree layers.

Our method was able to discover the equation

$$-0.715x^2 + 0.852xy - 0.715y^2 + 1.719 = 0$$

which can be simplified as

$$4.993x^2 - 5.947xy + 4.988y^2 + 12.000 = 0,$$

as desired. The graph of the learned equation is depicted overlayed with the true curve in the left side of Figure 5.7.

In the process of producing the true equation, many incorrect approximate equations were also generated. This naturally happens because the algorithm evaluated different

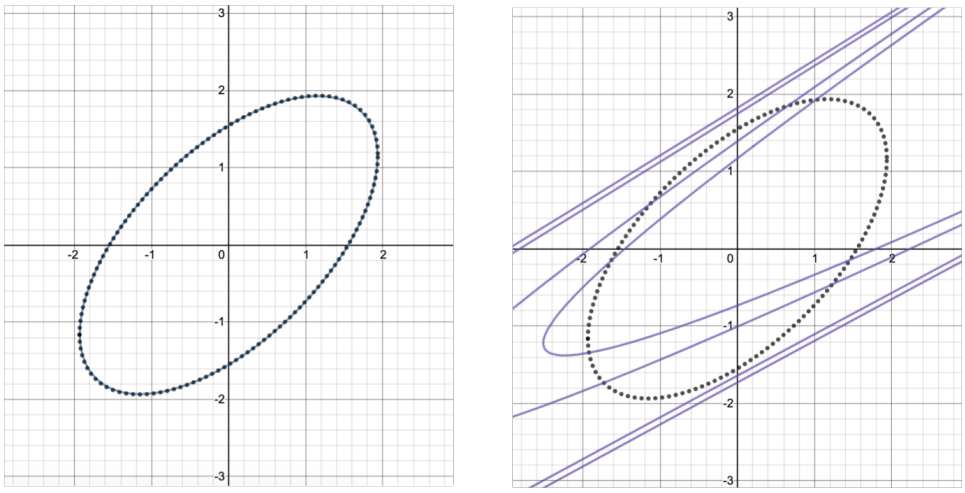

Figure 5.7: **Left:** the correctly learned implicit equation for the oblique ellipse. The dotted points are the true values and the smooth curve is the learned equation. **Right:** a sample incorrect equation obtained using our method, to illustrate the kind of implicitly defined equations that may be discovered in the process of finding the best one.

kinds of equations to find the rule to fit the data. One of the incorrect equations is

$$1.02\sqrt{\left|1.25\sqrt{|-0.02x+0.03y-0.05|}+1.41\sqrt{|0.02x-0.03y-0.04|}-0.53\right|}-0.12=0.$$

For illustration, the graph of this equation is depicted in the right side of Figure 5.7.

5.5 Summary

In this chapter, we showed how a symbolic function learner can be adapted to discover mathematical structure underlying a dataset in an unsupervised way. We showed how this method can be used to learn properties about structured datasets, demonstrating our technique by recovering some of the well-known laws of geometry with minimal human intervention.

The implications of this work are significant as it opens new pathways for scientific discovery in less time and using fewer resources. We note that the unsupervised discovery approach is not restricted to using our own SFL; it would be interesting to see how other

symbolic function learning algorithms would perform in the proposed system. Although our own SFL has shown success in several tasks, we hope to improve its capacity in terms of accuracy and scalability even further.

Chapter 6

Conclusions and Future Work

Above every man of knowledge
is one who knows better.

Noble Qur'an, 12:76

We bring this thesis to a close by highlighting the impact of our work and suggesting some directions for possible future research.

6.1 Impact of this Thesis

We set out with the goal of advancing the state of automating knowledge discovery using neural networks. In particular, we focused on designing an algorithm for learning formulas in symbolic mathematics, and presenting several ways in which the algorithm could be used to discover patterns and relationships between quantitative variables in the world we live in.

This thesis presented a novel symbolic function learner (SFL) based on ideas from neural networks to enable deep learning training capabilities. We showed how our SFL compares competently with existing models in regression tasks, without introducing biases or heuristics that restrict its application to problems in the domain of the natural sciences.

We also described a number of ways that symbolic function learners, including our own, can be used to learn mathematical relationships from measured datasets. These tasks include the discovery of differential equations as well as solving them. While methods for

both finding and solving differential equations have existed for a long time, our proposed system both makes use of deep learning and returns symbolic expressions, a combination that has not been commonly addressed in previous work. Our method is modular and can accommodate any symbolic function learner, providing a useful testing ground for evaluating other symbolic regression algorithms.

Furthermore, we presented a method for adapting a symbolic function learner to find mathematical relationships between measured quantities in an unsupervised way. This method does not explicitly label any variate as being an output value, bypassing the assumptions made in standard regression problems about causal relationships. We demonstrated how this method was able to recover rules of geometry from raw data alone. This included the law of sines and the Pythagorean theorem, two relationships on measurements relating to triangles where no variable is necessarily considered an output of a function of the others.

6.2 Future Research

The work presented in this thesis leads towards the following directions for future research.

Improving Scalability

The SFL we present in Chapter 3 would benefit from improvements in scalability, allowing it to more robustly handle situations with more input variables, more allowable operators, and more tree layers. One way this might be done is performing more extensive experiments to optimize hyperparameters such as initial weight distributions, learning rates, training protocols, regularization tactics, and others. It may also help to introduce more structural changes into the architecture itself.

Employing Alternative SFLs

The knowledge discovery tasks presented in this thesis, including the learning and solving of differential equations and unsupervised rules discovery, lend themselves well for accommodating symbolic function learners other than the SFL we introduce in Chapter 3. It is worth investigating how these methods might perform when provided other symbolic function learners, including ones based on genetic algorithms as well as other neural network approaches. This could also become a useful arena for evaluating the strength of symbolic function learners in other environments than the ones they were designed for.

Expanding Application Areas

The unsupervised rules learning method of Chapter 5 offers a lot of potential for scientific and mathematical discovery beyond the domain of geometry that was explored in this work. The presented method can be applied in other areas and there is great potential in the kind of formulas and relationships can be learned, recovered, and discovered in other domains.

6.3 Closing Remarks

Up until recently, little work has been done to incorporate neural networks with symbolic mathematics. Furthermore, symbolic function learners of all types have not found much use outside the task of symbolic regression. We hope that this thesis, with its contributions in symbolic function learning and related applications, will help advance both of these areas of study. In doing so, we hope that this work will contribute towards the progress of knowledge discovery as a whole.

References

[1] Ismail Alaoui Abdellaoui and Siamak Mehrkanoon. Symbolic regression for scientific discovery: an application to wind speed forecasting. *arXiv preprint arXiv:2102.10570*, 2021.

[2] G Adomian, R Rach, and NT Shawagfeh. On the analytic solution of the Lane-Emden equation. *Foundations of Physics Letters*, 8(2):161–181, 1995.

[3] Muhammad Aurangzeb Ahmad and Şener Özönder. Physics inspired models in artificial intelligence. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3535–3536, 2020.

[4] Baligh Al-Helali, Qi Chen, Bing Xue, and Mengjie Zhang. A new imputation method based on genetic programming and weighted KNN for symbolic regression with incomplete data. *Soft Computing*, pages 1–20, 2021.

[5] Nikola Andelić, Sandi Baressi Šegota, Ivan Lorencin, Vedran Mrzljak, and Zlatan Car. Estimation of COVID-19 epidemic curves using genetic programming algorithm. *Health Informatics Journal*, 27(1):1460458220976728, 2021.

[6] Forough Arabshahi, Zhichu Lu, Sameer Singh, and Animashree Anandkumar. Memory augmented recursive neural networks. *arXiv*, 2019.

[7] Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. Towards solving differential equations through neural programming.

[8] Dhananjay Ashok, Joseph Scott, Sebastian Wetzel, Maysum Panju, and Vijay Ganesh. Logic guided genetic algorithms. *arXiv preprint arXiv:2010.11328*, 2020.

[9] Douglas Adriano Augusto and Helio JC Barbosa. Symbolic regression via genetic programming. In *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*, pages 173–178. IEEE, 2000.

[10] Vernon Austel, Cristina Cornelio, Sanjeeb Dash, Joao Goncalves, Lior Horesh, Tyler Josephson, and Nimrod Megiddo. Symbolic regression using mixed-integer nonlinear optimization. *arXiv preprint arXiv:2006.06813*, 2020.

[11] Maxime Bassenne and Adrián Lozano-Durán. Computational model discovery with reinforcement learning. *arXiv preprint arXiv:2001.00008*, 2019.

[12] João Batista, Ana Cabral, Maria Vasconcelos, Leonardo Vanneschi, and Sara Silva. Improving land cover classification using genetic programming for feature construction. 2020.

[13] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[14] Jens Berg and Kaj Nyström. Data-driven discovery of PDEs in complex datasets. *Journal of Computational Physics*, 384:239–252, 2019.

[15] Luca Biggio, Tommaso Bendinelli, Aurelien Lucchi, and Giambattista Parascandolo. A seq2seq approach to symbolic regression. *1st Workshop on Learning Meets Combinatorial Algorithms @ NeurIPS 2020*, 2020.

[16] Lynne Billard and Edwin Diday. Symbolic regression analysis. In *Classification, Clustering, and Data Analysis*, pages 281–288. Springer, 2002.

[17] Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[18] Jure Brence, Ljupčo Todorovski, and Sašo Džeroski. Probabilistic grammars for equation discovery. *arXiv preprint arXiv:2012.00428*, 2020.

[19] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[20] Wenbo Cao and Weiwei Zhang. Machine learning of partial differential equations from noise data. *arXiv preprint arXiv:2010.06507*, 2020.

[21] Chen Chen, Changtong Luo, and Zonglin Jiang. Elite bases regression: A real-time algorithm for symbolic regression. In *2017 13th International Conference on Natural*

Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pages 529–535. IEEE, 2017.

[22] Gang Chen. Learning symbolic expressions via Gumbel-Max equation learner network. *arXiv preprint arXiv:2012.06921*, 2020.

[23] Qi Chen, Bing Xue, and Mengjie Zhang. Improving symbolic regression based on correlation between residuals and variables. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 922–930, 2020.

[24] Zhao Chen, Yang Liu, and Hao Sun. Deep learning of physical laws from scarce data. *arXiv preprint arXiv:2005.03448*, 2020.

[25] Hatice Citakoglu, Bilal Babayigit, and Nese Acanal Haktanir. Solar radiation prediction using multi-gene genetic programming approach. *Theoretical and Applied Climatology*, 142(3):885–897, 2020.

[26] Alison Cozad and Nikolaos V Sahinidis. A global MINLP approach to symbolic regression. *Mathematical Programming*, 170(1):97–119, 2018.

[27] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *arXiv preprint arXiv:2006.11287*, 2020.

[28] Miles D Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks. *arXiv preprint arXiv:1909.05862*, 2019.

[29] James W Davidson, Dragan A Savic, and Godfrey A Walters. Symbolic and numerical regression: experiments and applications. *Information Sciences*, 150(1-2):95–117, 2003.

[30] Ernest Davis. The use of deep learning for symbolic integration: A review of (Lample and Charton, 2019). *arXiv preprint arXiv:1912.05752*, 2019.

[31] Mehdi Dehghan and Mehdi Tatari. The use of He's variational iteration method for solving a Fokker-Planck equation. *Physica Scripta*, 74(3):310, 2006.

[32] Hannes Deponte, Alberto Tonda, Nathalie Gottschalk, Laurent Bouvier, Guillaume Delaplace, Wolfgang Augustin, and Stephan Scholl. Two complementary methods for the computational modeling of cleaning processes in food industry. *Computers & Chemical Engineering*, 135:106733, 2020.

[33] Romain Edelmann and Viktor Kunčak. Neural-network guided expression transformation. *arXiv preprint arXiv:1902.02194*, 2019.

[34] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

[35] Taylor Faucett, Jesse Thaler, and Daniel Whiteson. Mapping machine-learned physics into a human-readable space. *arXiv preprint arXiv:2010.11998*, 2020.

[36] Javier Garcia-Bernardo and Petr Janskỳ. Profit shifting of multinational corporations worldwide. 2021.

[37] Ahmed Gondia, Mohamed Ezzeldin, and Wael El-Dakhakhni. Mechanics-guided genetic programming expression for shear-strength prediction of squat reinforced concrete walls with boundary elements. *Journal of Structural Engineering*, 146(11):04020223, 2020.

[38] Seungwoong Ha and Hawoong Jeong. Discovering conservation laws from trajectories via machine learning. *arXiv preprint arXiv:2102.04008*, 2021.

[39] Nguyen Thi Hien, Cao Truong Tran, Xuan Hoai Nguyen, Sooyoul Kim, Vu Dinh Phai, Nguyen Ba Thuy, and Ngo Van Manh. Genetic programming for storm surge forecasting. *Ocean Engineering*, 215:107812, 2020.

[40] Zhilong Huang, Yanping Tian, Chunjiang Li, Guang Lin, Lingling Wu, Yong Wang, and Hanqing Jiang. Data-driven automated discovery of variational laws hidden in physical systems. *Journal of the Mechanics and Physics of Solids*, 137:103871, 2020.

[41] Ilknur Icke and Joshua C Bongard. Improving genetic programming based symbolic regression using deterministic machine learning. In *2013 IEEE Congress on Evolutionary Computation*, pages 1763–1770. IEEE, 2013.

[42] Raban Iten, Tony Metger, Henrik Wilming, Lídia Del Rio, and Renato Renner. Discovering physical concepts with neural networks. *Physical Review Letters*, 124(1):010508, 2020.

[43] Yuma Iwasaki and Masahiko Ishida. Data-driven formulation of natural laws by recursive-LASSO-based symbolic regression. *arXiv preprint arXiv:2102.09210*, 2021.

[44] Miguel Jaques, Michael Burke, and Timothy Hospedales. Physics-as-inverse-graphics: Joint unsupervised learning of objects and physics from video. *arXiv preprint arXiv:1905.11169*, 2019.

[45] Kadierdan Kaheman, J Nathan Kutz, and Steven L Brunton. SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics. *Proceedings of the Royal Society A*, 476(2242):20200279, 2020.

[46] S Kazem, JA Rad, and K Parand. Radial basis functions methods for solving Fokker–Planck equation. *Engineering Analysis with Boundary Elements*, 36(2):181–189, 2012.

[47] Alireza Khanteymoori, Fatemeh Alamdar, and Farzaneh Ghorbani. ARP: asexual reproduction programming. *Connection Science*, pages 1–22, 2020.

[48] Joanne T Kim, Sookyung Kim, and Brenden K Petersen. An interactive visualization platform for deep symbolic regression. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020.

[49] Jongeun Kim, Sven Leyffer, and Prasanna Balaprakash. Learning symbolic expressions: Mixed-integer formulations, cuts, and heuristics. *arXiv preprint arXiv:2102.08351*, 2021.

[50] Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, and Marin Soljačić. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[51] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, pages 1–31, 2019.

[52] Robert Kragler. On mathematica program for "poor man's integrator" implementing Risch-Norman algorithm. *Programming and Computer Software*, 35(2):63–78, 2009.

[53] Gabriel Kronberger, J Manuel Colmenar, Stephan M Winkler, and J Ignacio Hidalgo. Multilayer analysis of population diversity in grammatical evolution for symbolic regression. *Soft Computing*, 24(15):11283–11295, 2020.

[54] Jiří Kubalík, Erik Derner, and Robert Babuška. Symbolic regression driven by training data and prior knowledge. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 958–966, 2020.

[55] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[56] Mehrdad Lakestani and Mehdi Dehghan. Numerical solution of Fokker-Planck equation using the cubic B-spline scaling functions. *Numerical Methods for Partial Differential Equations: An International Journal*, 25(2):418–429, 2009.

[57] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *International Conference on Learning Representations*, 2020.

[58] Li Li, Minjie Fan, Rishabh Singh, and Patrick Riley. Neural-guided symbolic regression with asymptotic constraints. *arXiv preprint arXiv:1901.07714*, 2019.

[59] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[60] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[61] Hidde Lingmont. Data-driven techniques for finding governing equations of noisy nonlinear dynamical systems. 2020.

[62] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, and Gary Yen. A survey on evolutionary neural architecture search. *arXiv preprint arXiv:2008.10937*, 2020.

[63] Lennart Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

[64] Zichao Long, Yiping Lu, and Bin Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

[65] Xuan Ma, Xing Li, Rong-Jun Tang, and Qing Liu. A particle swarm optimization approach for symbolic regression. *Acta Automatica Sinica*, 46(8):1714–1726.

[66] Afsaneh Mahanipour and Hossein Nezamabadi-Pour. GSP: an automatic programming technique with gravitational search algorithm. *Applied Intelligence*, 49(4):1502–1516, 2019.

[67] Shehryar Malik, Usman Anwar, Ali Ahmed, and Alireza Aghasi. Learning to solve differential equations across initial conditions. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[68] Jorge Martinez-Gil and Jose M Chaves-Gonzalez. A novel method based on symbolic regression for interpretable semantic similarity measurement. *Expert Systems with Applications*, 160:113663, 2020.

[69] Georg Martius and Christoph H Lampert. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, 2016.

[70] Mikhail Maslyaev and Alexander Hvatov. Multi-objective discovery of PDE systems using evolutionary approach. *arXiv preprint arXiv:2103.06739*, 2021.

[71] Ben McKay, Mark J Willis, and Geoffrey W Barton. Using a tree structured genetic algorithm to perform symbolic regression. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, pages 487–492. IET, 1995.

[72] Andrew J Meade Jr and Alvaro A Fernandez. Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 20(9):19–44, 1994.

[73] Renato Miranda Filho, Anisio Lacerda, and Gisele L Pappa. Explaining symbolic regression predictions. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[74] Pierluigi Morano, Francesco Tajani, Marco Locurcio, Felicia Di Liddo, and Debora Anelli. An application to a Spanish case study of a property valuation models. In *Appraisal and Valuation*, pages 79–90. Springer, 2021.

[75] Joel Moses. Symbolic integration: the stormy decade. *Communications of the ACM*, 14(8):548–560, 1971.

[76] A Murari, E Peluso, M Gelfusa, I Lupelli, M Lungaroni, and P Gaudio. Symbolic regression via genetic programming for data driven derivation of confinement scaling laws without any assumption on their mathematical form. *Plasma Physics and Controlled Fusion*, 57(1):014008, 2014.

[77] Miguel Nicolau and Alexandros Agapitos. Choosing function sets with better generalisation performance for symbolic regression models. *Genetic Programming and Evolvable Machines*, pages 1–28, 2020.

[78] Luiz Otavio VB Oliveira, Joao Francisco BS Martins, Luis F Miranda, and Gisele L Pappa. Analysing symbolic regression benchmarks under a meta-learning approach.

In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1342–1349, 2018.

[79] Patryk Orzechowski, William La Cava, and Jason H Moore. Where are we now? A large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1183–1190, 2018.

[80] K Parand, Mehdi Dehghan, AR Rezaei, and SM Ghaderi. An approximation algorithm for the solution of the nonlinear Lane–Emden type equations arising in astrophysics using hermite functions collocation method. *Computer Physics Communications*, 181(6):1096–1108, 2010.

[81] Daniel R Parisi, Marıa C Mariani, and Miguel A Laborde. Solving differential equations with unsupervised neural networks. *Chemical Engineering and Processing: Process Intensification*, 42(8-9):715–721, 2003.

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

[83] Brenden K Petersen. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

[84] Tong Qin, Kailiang Wu, and Dongbin Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.

[85] Markus Quade, Markus Abel, Kamran Shafi, Robert K Niven, and Bernd R Noack. Prediction of dynamical systems by symbolic regression. *Physical Review E*, 94(1):012214, 2016.

[86] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[87] Chengping Rao, Hao Sun, and Yang Liu. Physics informed deep learning for computational elastodynamics without labeled data. *arXiv preprint arXiv:2006.08472*, 2020.

[88] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

[89] Hannes Risken. Fokker-planck equation. In *The Fokker-Planck Equation*, pages 63–95. Springer, 1996.

[90] Syed Muhammad Sajjad Rizvi. *On the Importance of Infrastructure-Awareness in Large-Scale Distributed Storage Systems*. Doctoral thesis, University of Waterloo, 2021.

[91] R Rueda, Luis G Baca Ruíz, Manuel P Cuéllar, and MC Pegalajar. An ant colony optimization approach for symbolic regression using straight line programs. application to energy consumption modelling. *International Journal of Approximate Reasoning*, 121:23–38, 2020.

[92] Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pages 4439–4447, 2018.

[93] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[94] Joseph Scott, Maysum Panju, and Vijay Ganesh. LGML: Logic guided machine learning (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13909–13910, 2020.

[95] Hassam Sheikh, Shauharda Khadka, Santiago Miret, and Somdeb Majumdar. Learning intrinsic symbolic rewards in reinforcement learning. *arXiv preprint arXiv:2010.03694*, 2020.

[96] David Speiser and Kim Williams. *Discovering the principles of mechanics 1600-1800: essays by David Speiser*, volume 1. Springer Science & Business Media, 2008.

[97] Waad Subber, Piyush Pandita, Sayan Ghosh, Steven Atkinson, Genghis Khan, Liping Wang, and Roger Ghanem. Data-based discovery of governing equations. *arXiv preprint arXiv:2012.06036*, 2020.

[98] Jovan Tanevski, Ljupčo Todorovski, and Sašo Džeroski. Combinatorial search for selecting the structure of models of dynamical systems with equation discovery. *Engineering Applications of Artificial Intelligence*, 89:103423, 2020.

[99] Maksimilian Tarasevich, Aleksei Tepljakov, Eduard Petlenkov, and Vitali Vansovits. Modeling and identification of an industrial hot water boiler. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pages 285–290. IEEE, 2020.

[100] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[101] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

[102] Sohrab Towfighi. Symbolic regression by uniform random global search. *SN Applied Sciences*, 2(1):1–11, 2020.

[103] Mike G Tsionas and A George Assaf. Symbolic regression for better specification. *International Journal of Hospitality Management*, 91:102638, 2020.

[104] Ioannis G Tsoulos, Dimitris Gavrilis, and Euripidis Glavas. Solving differential equations with constructed neural networks. *Neurocomputing*, 72(10-12):2385–2391, 2009.

[105] Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *arXiv preprint arXiv:2006.10782*, 2020.

[106] Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16), 2020.

[107] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, Robert I McKay, and Edgar Galván-López. Semantically-based crossover in genetic programming: Application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

[108] Francisco Villaescusa-Navarro, Daniel Anglés-Alcázar, Shy Genel, David N Spergel, Rachel S Somerville, Romeel Dave, Annalisa Pillepich, Lars Hernquist, Dylan Nelson, Paul Torrey, et al. The CAMELS project: Cosmology and Astrophysics with MachinE Learning Simulations. *arXiv preprint arXiv:2010.00619*, 2020.

[109] Yiqun Wang, Nicholas Wagner, and James M Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, 2019.

[110] Sebastian J Wetzel, Roger G Melko, Joseph Scott, Maysum Panju, and Vijay Ganesh. Discovering symmetry invariants and conserved quantities by interpreting siamese neural networks. *arXiv preprint arXiv:2003.04299*, 2020.

[111] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. *arXiv preprint arXiv:2003.04919*, 2020.

[112] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

[113] Tailin Wu and Max Tegmark. Toward an artificial intelligence physicist for unsupervised learning. *Physical Review E*, 100(3):033311, 2019.

[114] Hengrui Xing, Ansaf Salleb-Aouissi, and Nakul Verma. Automated symbolic law discovery: A computer vision approach. 2021.

[115] Ramu Yerukala and Naveen Kumar Boiroju. Approximations to standard normal distribution function. *International Journal of Scientific & Engineering Research*, 6(4):515–518, 2015.

[116] Jan Žegklitz and Petr Pošík. Benchmarking state-of-the-art symbolic regression algorithms. *Genetic Programming and Evolvable Machines*, pages 1–29, 2020.

[117] Zhiming Zhang and Yongming Liu. Robust data-driven discovery of partial differential equations under uncertainties. *arXiv preprint arXiv:2102.06504*, 2021.

[118] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.