Towards Solving Differential Equations through Neural Programming

Forough Arabshahi [1] Sameer Singh [1] Animashree Anandkumar [2]

1. Introduction

Differential equations are used to model numerous phenomena such as heat, electrodynamics, fluid dynamics and quantum mechanics. For example, Partial Differential Equations (PDEs) have been used for boundary control of robotic aircrafts (Paranjape et al., 2013), control of autonomous agents (Bandyopadhyay et al., 2017) or generating fluid animations (Yang et al., 2016). Although some differential equations have easy to find solutions, such differential equations do not usually emerge in real-world problems. Therefore, solving differential equations is often a bottleneck in real-world applications. It is important to solve them accurately, in a timely and cost-effective manner.

Researchers have been interested in solving differential equations with neural networks ever since the beginning of the 90's (Meade Jr & Fernandez, 1994). Almost all of them focus on gathering *numerical evaluations* from the differential equation and using this data for training a neural network that interpolates/approximates the solution (Lagaris et al., 1998; Rudd & Ferrari, 2015; Berg & Nyström, 2017b; Trischler & D'Eleuterio, 2016; Berg & Nyström, 2017a; Sirignano & Spiliopoulos, 2017; Han et al., 2017; Khoo et al., 2018). Although intuitive, we argue that this approach is not scalable and will require training and tuning a separate neural network for each problem. We propose an alternative method in this extended abstract in the hope of initiating a new direction for tackling this problem.

Our main proposal is to use *symbolic* data to train a neural model for solving a differential equation. Symbolic equations provide an efficient and compact representation of the differential equation and we leverage their compositionality to efficiently train a neural model capable of solving a differential equation in a scalable and generalizable manner.

We divide solving a differential equation into two main steps: (1) finding a set of candidate solutions to the differential equation, (2) accepting the correct solution from the given set of candidates using a neural model. Step (1) has been addressed in other contexts (Zaremba et al., 2014; Vijayakumar et al., 2018; Polosukhin & Skidanov, 2018) and these methods can be adapted to address our problem. Therefore, we focus instead on step (2) in this extended abstract.

[1]UC Irvine [2]California Institute of Technology. Correspondence to: Forough Arabshahi <farabsha@uci.edu>.

Published at the ICML workshop Neural Abstract Machines & Program Induction v2 (NAMPI) — Extended Abstract, Stockholm, Sweden, 2018. Copyright 2018 by the author(s).

Summary of Contributions We propose using *symbolic* data for training neural networks that solve differential equations. This results in a generalizable and scalable neural solver. The main reason is that we jointly learn a large number of functions, that cover an entire mathematical domain, and use these trained functions for solving an unseen differential equation. Almost all of the literature focuses on hand-crafting architectures that are tailored for a specific type of differential equation. Moreover, they use numerical evaluations of a differential equation for training, which means that training and tuning needs to be redone for solving a different input differential equation resulting in a lack of scalability and generalizability.

In this work, we investigate the possibility of using neural programs for solving ordinary differential equations (ODEs) by verifying/rejecting a candidate solution of an ODE. We design a neural programmer that is capable of choosing the correct solution with a high accuracy. Our neural programmer, based on a Tree-LSTM (Tai et al., 2015), leverages the compositionality of each input ODE.

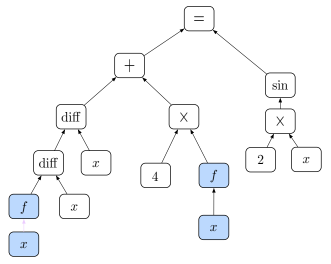
2. Problem Description

An n^{th}-order ordinary differential equation is of the following general form $\sum_{i=0}^{n} a_i(x)\frac{\mathrm{d}^i f(x)}{\mathrm{d}x^i} = b(x)$ where $a_i(x)$ for $i = 0, 1, \dots, n$ and $b(x)$ are arbitrary differentiable functions. $f(x)$ is an unknown function and $\frac{\mathrm{d}^i f(x)}{\mathrm{d}x^i}$ is its i^{th} derivative. Solving an ODE amounts to finding $f(x)$ that satisfies the general form equation.

Our goal is to take a step towards solving ODEs through neural programming. Formally, Given a candidate solution to the differential equation, we would like to verify whether it is the right solution or not.

3. Solving Differential Equations

In this extended abstract, we represent the problem of solving differential equations as a two-step process. In step 1, one finds a set of candidate solutions for the differential equation; given that the correct solution lies in the set, in step 2, the correct solution is chosen from the provided set.



(a) $\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} + 4f(x) = \sin(2x)$

Figure 1. Example of an ODE given in Fig. 1a. The solution of this ODE is $f(x) = \frac{1}{8}\sin(2x) - \frac{x}{4}\cos(2x)$. If the ODE has a closed form solution, $f(x)$ has a compositionality and will replace the blue nodes in the figure.

Table 1. Statistics of the data

Statistics	Sym	fEval	ODE
#data: train	13,375	1,760	7,071
#data: validation	1,477	641	793
#data: test	3,723	1,041	1,945
Min Depth	1	2	3
Max Depth	7	4	7
Average Depth	3.14	3.07	6.82

Since step 1 has been addressed in the literature, we present our approach for solving step 2.

We collect a large dataset of symbolic ODEs along with a set of correct and incorrect solutions for each. We augment this data with a symbolic dataset of equations that encode the properties of the functions appearing in the ODEs as well as a limited number of function evaluation data for the same functions. The data generation process is described in detail in Sec 4 and the model that uses this data is described below. We will show in Sec 4, that we are able to accurately accept or reject the correct or incorrect solution, respectively.

Proposed Compositional Model Our model leverages the compositionality of the input equations. We propose using tree-LSTMs that mirrors the structure of each input equation. Tree-LSTMs, proposed by (Tai et al., 2015) have shown good performance modeling mathematical equations (Arabshahi et al., 2018). Therefore, we use Tree-LSTMs for solving differential equations.

4. Experiments and Results

Dataset Our dataset contains symbolic ODEs and a set of candidate correct and incorrect solutions for each. It also contains the rules of differentiation for the functions in the dataset. This data is generated through the method proposed by Arabshahi et al. (2018). The data is further augmented with the dataset released by the same authors, which we refer to as the *mathData*. The properties of the final dataset used here is given in Table 1

Baseline Models We compare Tree LSTMs with Sympy, and simple Tree-structured neural networks (TreeNNs).

Table 2. Performance evaluation for solving ODEs on unseen test data: *MSE* is the mean squared error for the numeric data, *SymAcc* is accuracy of symbolic data not containing ODEs, *ODEacc* is the accuracy of the ODE data. Finally, *Acc* is the weighted average of SymAcc and ODEacc. Sym, Sym+ODE and full refer to the data used for training in each experiment.

Approach	Acc	MSE	SymAcc	ODEacc
Majority Class	52.15	-	50.16	56.45
Sympy	53.42	-	80.07	59.78
TreeNN sym	92.35	-	92.35	-
TreeLSTM sym	96.43	-	**96.43**	-
TreeNN ODE	98.45	-	-	98.45
TreeLSTM ODE	99.27	-	-	**99.27**
TreeNN sym+ODE	93.99	-	91.42	98.92
TreeLSTM sym+ODE	96.73	-	95.86	98.41
TreeNN full	93.49	7.59	90.61	99.02
TreeLSTM full	95.58	**0.051**	94.09	98.45

Since Sympy was used for generating correct equations and solutions to the ODE, it will always predict the correct solution or a correct identity. Sympy's loss in performance is the result of not being able to reject an incorrect solution to the ODE or an incorrect equation. We use different datasets to train TreeLSTMs and TreeNNs. Namely, *sym*, *ODE*, *sym+ODE* and *full*, mean that the models are trained with symbolic data, ODE data, symbolic and ODE data, and finally symbolic, ODE and numeric data combined.

Results Our model is implemented using MxNet (Chen et al., 2015). We tune all the models over the same grid of parameters and report the best for each model.

An important point is that the depth of ODE equations is at least 4 with an average of 6. SOTA has shown performance on up to depth 4 and the fact that the model is able to accurately accept/reject the solutions is a promising result for future research in this direction. The results are presented in Table 2. As shown, we achieve up to 99.27% accuracy for predicting the correct answer for a given ODE. Since column **Acc** shows the performance on different test samples for different models, we do not highlight any row.

5. Conclusions and Future Directions

We present a novel approach for solving differential equations and take a step towards solving them. Instead of gathering numeric data from ODEs for training, we propose using symbolic ODEs, resulting in generalizable and scalable models. Our results indicate that there is a potential for solving differential equations using neural programs.

We follow two main goals in the future. First, we plan to extend the domain by adding partial differential equations. Second, we are working on an efficient search algorithm that outputs candidate solutions for the differential equations. This will complete solving differential equations using neural programming.

Acknowledgments

The authors would like to thank Amazon Inc., for the AWS credits. F. Arabshahi is supported by DARPA Award D17AP00002. A. Anandkumar is supported by Microsoft Faculty Fellowship, NSF CAREER Award CCF-1254106, DARPA Award D17AP00002 and Air Force Award FA9550-15-1-0221. S. Singh would like to thank Adobe Research and FICO for supporting this research.

References

Arabshahi, Forough, Singh, Sameer, and Anandkumar, Animashree. Combining symbolic expressions and black-box function evaluations for training neural programs. In *International Conference on Learning Representations (ICLR)*, 2018.

Bandyopadhyay, Saptarshi, Chung, Soon-Jo, and Hadaegh, Fred Y. Probabilistic and distributed control of a large-scale swarm of autonomous agents. *IEEE Transactions on Robotics*, 33(5):1103–1123, 2017.

Berg, Jens and Nyström, Kaj. Neural network augmented inverse problems for pdes. *arXiv preprint arXiv:1712.09685*, 2017a.

Berg, Jens and Nyström, Kaj. A unified deep artificial neural network approach to partial differential equations in complex geometries. *arXiv preprint arXiv:1711.06464*, 2017b.

Chen, Tianqi, Li, Mu, Li, Yutian, Lin, Min, Wang, Naiyan, Wang, Minjie, Xiao, Tianjun, Xu, Bing, Zhang, Chiyuan, and Zhang, Zheng. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. 2015.

Han, Jiequn, Jentzen, Arnulf, et al. Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning. *arXiv preprint arXiv:1707.02568*, 2017.

Khoo, Yuehaw, Lu, Jianfeng, and Ying, Lexing. Solving for high dimensional committor functions using artificial neural networks. *arXiv preprint arXiv:1802.10275*, 2018.

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

Meade Jr, Andrew J and Fernandez, Alvaro A. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 19(12):1–25, 1994.

Paranjape, Aditya A, Guan, Jinyu, Chung, Soon-Jo, and Krstic, Miroslav. Pde boundary control for flexible articulated wings on a robotic aircraft. *IEEE Transactions on Robotics*, 29(3):625–640, 2013.

Polosukhin, Illia and Skidanov, Alexander. Neural program search: Solving programming tasks from description and examples. *arXiv preprint arXiv:1802.04335*, 2018.

Rudd, Keith and Ferrari, Silvia. A constrained integration (cint) approach to solving partial differential equations using artificial neural networks. *Neurocomputing*, 155: 277–285, 2015.

Sirignano, Justin and Spiliopoulos, Konstantinos. Dgm: A deep learning algorithm for solving partial differential equations. *arXiv preprint arXiv:1708.07469*, 2017.

Tai, Kai Sheng, Socher, Richard, and Manning, Christopher D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Trischler, Adam P and D'Eleuterio, Gabriele MT. Synthesis of recurrent neural networks for dynamical system simulation. *Neural Networks*, 80:67–78, 2016.

Vijayakumar, Ashwin J, Mohta, Abhishek, Polozov, Oleksandr, Batra, Dhruv, Jain, Prateek, and Gulwani, Sumit. Neural-guided deductive search for real-time program synthesis from examples. *International Conference on Learning Representations (ICLR)*, 2018.

Yang, Cheng, Yang, Xubo, and Xiao, Xiangyun. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

Zaremba, Wojciech, Kurach, Karol, and Fergus, Rob. Learning to discover efficient mathematical identities. In *Advances in Neural Information Processing Systems*, pp. 1278–1286, 2014.

A. Stress tests

Test 1. depth analyzing the depth generalization

Table 3. trained on equations of depth 1,2,3

test depth	acc	prec	recall	MSE	sympy acc	pos/total	#samples
4	94.70	92.95	96.31	3.57E-06	71.93	45.58	2867
5	89.06	90.63	92.80	1.43E-09	77.77	60.73	4314
6	83.12	86.64	94.86	8.64E-38	76.50	68.19	1374