Towards Solving Differential Equations through Neural Programming

Forough Arabshahi* Sameer Singh* Animashree Anandkumar†

*University of California, Irvine †California Institute of Technology

Differential Equations

- Used to model numerous phenomena
  - heat
  - electrodynamics
  - fluid dynamics
  - quantum mechanics
  - ...
- An n^{th}-order ordinary differential equation (ODE)

$$a_0(x)f(x) + a_1(x)\frac{\mathrm{d}f(x)}{\mathrm{d}x} + \cdots + a_n(x)\frac{\mathrm{d}^n f(x)}{\mathrm{d}x^n} = b(x)$$

  - d is the differentiation operator
- Solving the differential equation: find $f(x)$ that satisfies it
- Not always easy to find solutions to differential equations
- Can neural networks be used to solve differential equations?
  - Researchers have been addressing this since the 90's
    - Gather numerical evaluations of a given differential equation
    - Use the evaluations to interpolate/approximate the solution with a neural network
- Drawbacks
  - Hand tailored architecture for each differential equation
  - Train and tune a neural network for each differential equation
  - Lack of scalability and generalizability

- Goal: Find a generalizable and scalable solution
- How?
  - Use symbolic differential equations instead of numerical evaluations
  - Leverage the compositionality of the differential equation
- Proposed method:
  - Step 1: Find a set of candidate solutions to the differential equation
    - has been studied in the literature
  - Step 2: Choose the correct solution among the candidates
    - The focus of this work

Modeling Differential Equations

- Grammar rules:

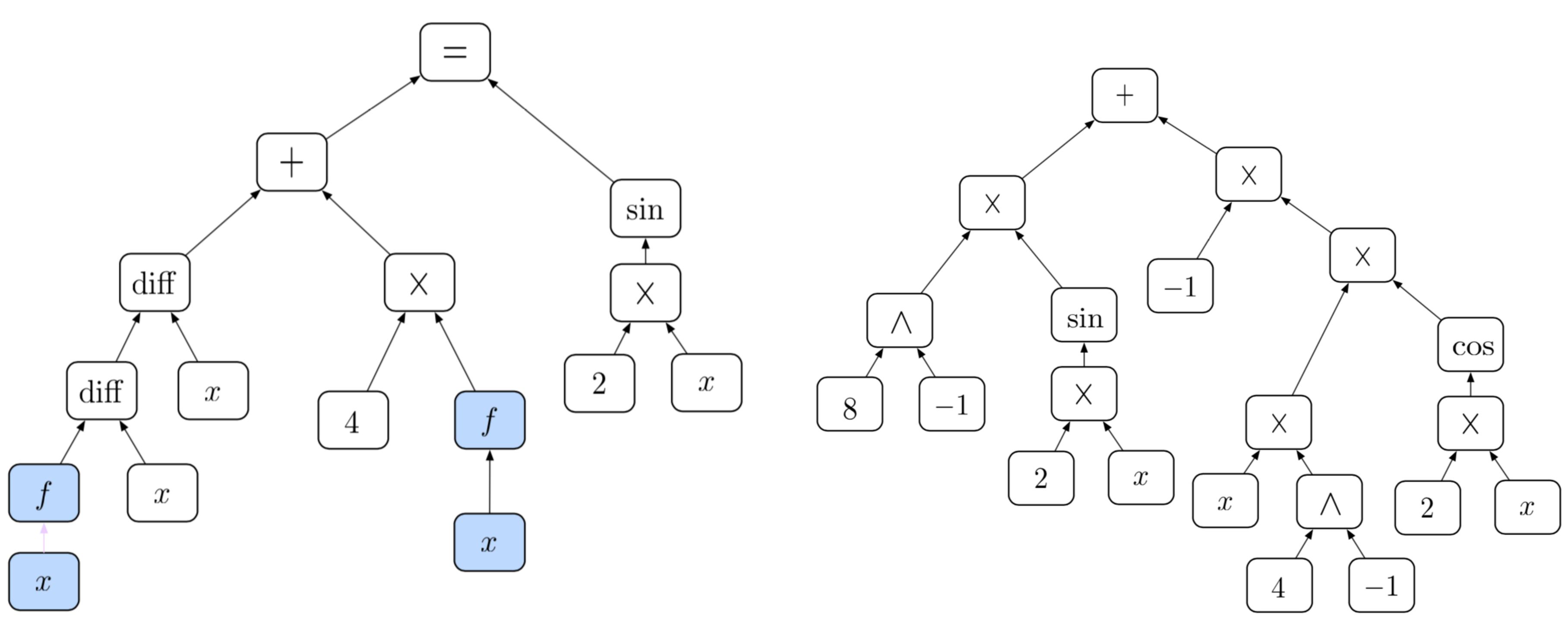
$$\begin{aligned}
&I \to =(E,E),\ \neq(E,E)\\
&E \to T, F_1(E), F_2(E,E)\\
&F_1 \to \sin, \cos, \tan, \ldots\\
&F_2 \to +, \wedge, \times, \text{diff}, \ldots\\
&T \to -1, 0, 1, 2, \pi, x, y, \ldots, \text{floating point numbers}
\end{aligned}$$

- Covered domain:

Table: Symbols in our grammar, i.e. the functions, variables, and constants

Unary functions, F_1					Terminal, T		Binary, F_2
sin	cos	csc	sec	tan	0	1	$+$
cot	arcsin	arccos	arccsc	arcsec	2	3	$\times$
arctan	arccot	sinh	cosh	csch	4	10	$\wedge$
sech	tanh	coth	arsinh	arcosh	0.5	-1	diff
arcsch	arsech	artanh	arcoth	exp	0.4	0.7	
					π	x	

- Example of a differential equation and its solution



an ODE $\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} + 4f(x) = \sin(2x)$ Its solution $f(x): \frac{1}{8}\sin(2x) - \frac{x}{4}\cos(2x)$

Choosing the correct candidate solution

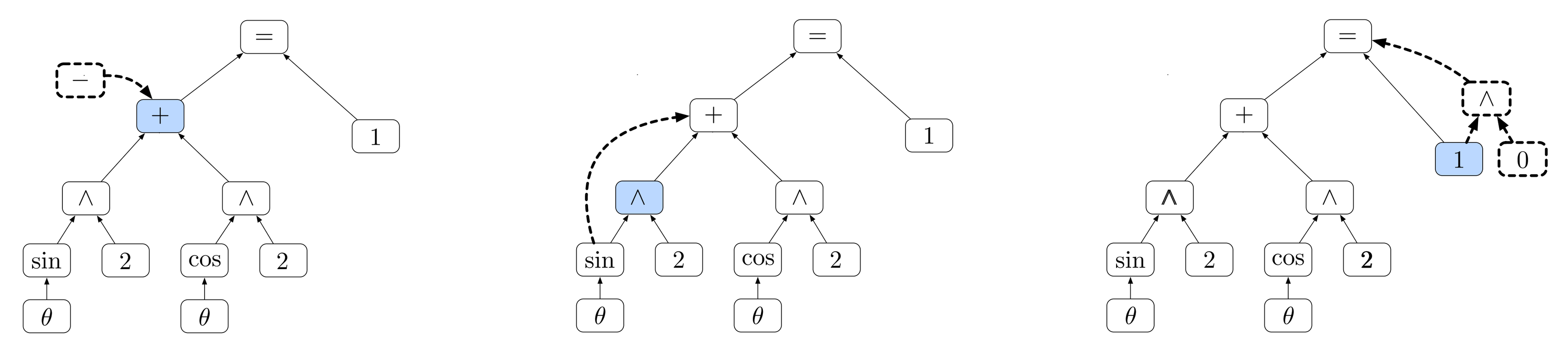
- Tree LSTM whose structure mirrors the input equation
  - LSTM cells associated with each Function
  - 1-layer feed-forward net for embedding symbolic terminals
  - 2-layer feed-forward net for encoding floating point numbers
  - 2-layer feed-forward net for decoding floating point numbers
- Data used in training:
  - symbolic identities that express the relationships between functions (sym)
  - symbolic ODEs and a set of candidate solutions for each (ODE)
  - single function numeric evaluations (num)

- Baselines:
  - Majority class
  - Sympy
  - Tree-structured RNNs

Dataset Generation Scheme:

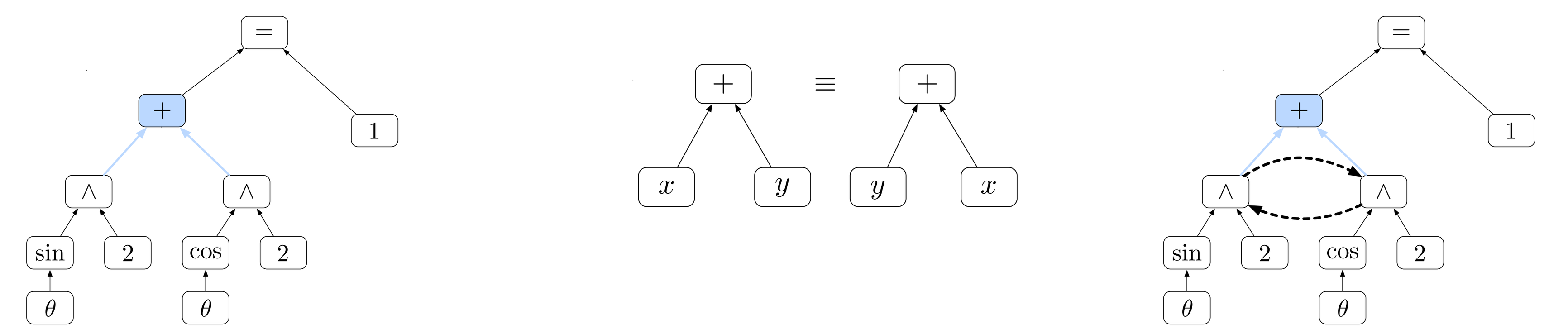
Generating Symbolic Equations

- Generate possible equations valid in the grammar
  - Start from a small initial set of axioms e.g. $\sin^2(\theta) + \cos^2(\theta) = 1$
  - For each axiom, choose a random tree node
  - Make local random changes to the node.



Replace Node Shrink Node Expand Node

  - Problem: More incorrect equations than correct
- Generate additional correct equations
  - Solution: Sub-tree matching with a *mathDictionary*



Choose a Node mathDictionary match value

Generating ODEs

- Randomly generate ODE coefficients
- Solve using Sympy
- If any solution:
  - Add to candidate solutions
  - Locally change the correct solution to generate incorrect candidates

Table: 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

Experiments and Results

Table: 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 (symbolic, symbolic+ODE and all the data, respectively) 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