# Neural-guided equation discovery

Jannis Brugger, David Richter, Mattia Cerrato, Mira Mezini, Stefan Kramer

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

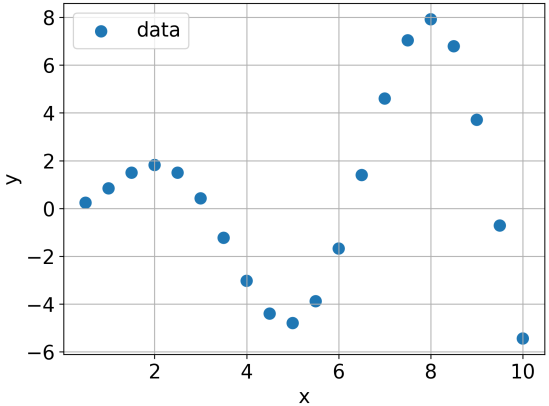
JG|U

3AI THE THIRD WAVE OF AI

TECHNISCHE UNIVERSITÄT DARMSTADT

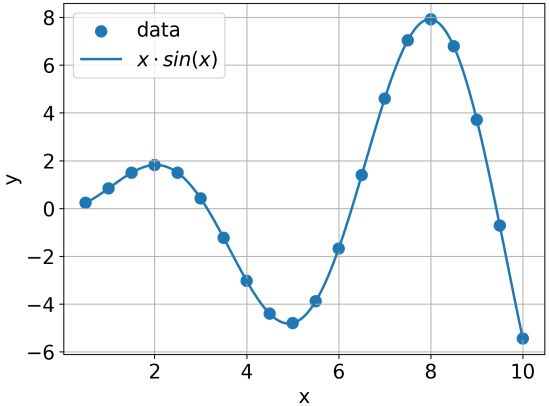
# Symbolic regression

- Problem of learning a symbolic expression from numerical data



# Symbolic regression

- Problem of learning a symbolic expression from numerical data



# Why equations?

- Most common form of knowledge in science
- Captures the relationships between features
- Understandable for humans

$$E_{kinetic} = \frac{1}{2} m \cdot v^2$$

# Why equations?

- Most common form of knowledge in science
- Captures the relationships between features
- Understandable for humans

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \cdot \sum_{n=0}^{\infty} \frac{(4n)! \cdot (1103 + 26390n)}{(n!)^4 \cdot 396^{4n}}$$

[J. Borwein, P. Borwein, and Bailey 1989]

# Contribution

Combining AlphaZero [Silver et al. 2018] and Dreamcoder [Ellis et al. 2021] for equation discovery

Important aspects

- Equation transfer learning
- Usage of context-free grammar
- Embedding of syntax trees and datasets

# Overview

**Input into system**

**Input into discovery of one equation**

**Expand one node in syntax tree**

**Output of discovery of one equation**

**Output of system**

# System view

**Input into system**

Dataset 1

| x | y |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

…

Dataset n

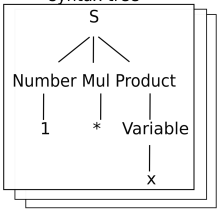
| x | y |
|---|---|
| 1 | 1 |
| 2 | 4 |
| 3 | 16 |

Untrained equation finder

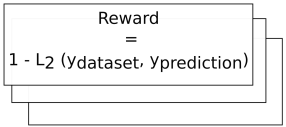
Grammar:
S → Number Plus Sum
S → Number Mul Product
…
Variable → x

**Output of system**

k complete syntax tree




Reward for k complete syntax trees




Trained equation finder

# Example datasets

Processed by

- Multilayer perceptron
- LSTM
- Bidirectional LSTM

Raw or normalized data

$y = x^2$

| x | y |
|---|---|
| 0.5 | 0.25 |
| 1 | 1 |
| ... | ... |
| 10 | 100 |

$y = log(x)$

| x | y |
|---|---|
| 0.5 | -0.3 |
| 1 | 0 |
| ... | ... |
| 10 | 1 |

$y = \frac{1}{x}$

| x | y |
|---|---|
| 0.5 | 2 |
| 1 | 1 |
| ... | ... |
| 10 | 0.1 |

# Grammar

- Usage of domain knowledge
- Start symbol $\langle S \rangle$
- Production rules
  - left $::=$ right

Generated equations:

$$(3 + x)$$
$$8$$
$$(9 + (1 * x)))$$

**Grammar**

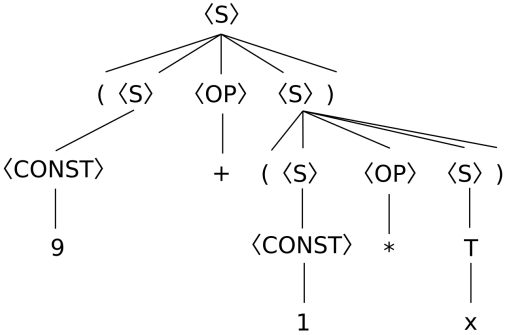
$\langle S \rangle ::= ( \langle S \rangle \ \langle OP \rangle \ \langle S \rangle \ ) | \langle T \rangle | \langle CONST \rangle$

$T ::= x | y$

$\langle OP \rangle \ ::= * | / | + | -$

$\langle CONST \rangle ::= 0 | 1 | ... | 9$

# Syntax tree $(9 + (1 * x))$




## Grammar

$\langle S \rangle ::= ( \langle S \rangle \langle OP \rangle \langle S \rangle ) | \langle T \rangle | \langle CONST \rangle$

$T ::= x | y$

$\langle OP \rangle ::= * | / | + | -$

$\langle CONST \rangle ::= 0 | 1 | ... | 9$

# Representation of syntax tree

- Hashed path [Alon et al. 2019]

  Pattern:

  left symbol , Hash(path), right symbol

  Path A :

  $\langle CONST \rangle, Hash(\langle S \rangle, \langle S \rangle \langle OP \rangle), +$

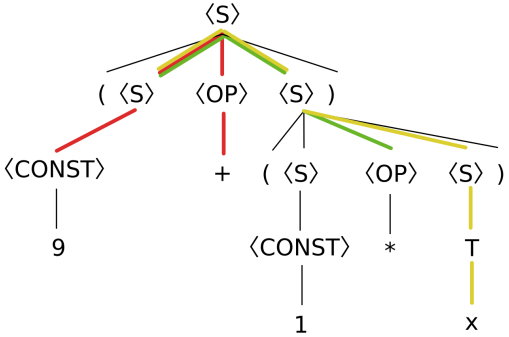
  Path B :

  $\langle S \rangle, Hash(\langle S \rangle \langle S \rangle), \langle OP \rangle$

  Path C :

  $\langle S \rangle, Hash(\langle S \rangle, \langle S \rangle, \langle S \rangle, \langle T \rangle), x$

  Path ...



# Representation of syntax tree

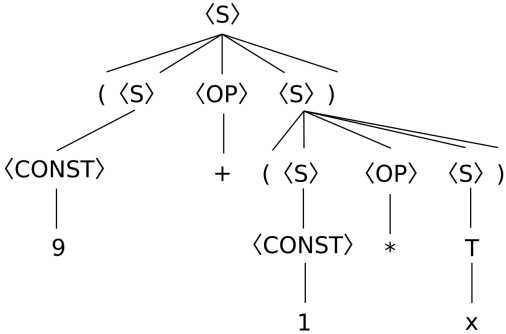
- Hashed path [Alon et al. 2019]
- Full tree

  Pattern:

  Node [Child Nodes]

  Example tree

  $\langle S \rangle [(\langle S \rangle [\ldots] \langle OP \rangle [+] \langle S \rangle [\ldots])]$



# Representation of syntax tree

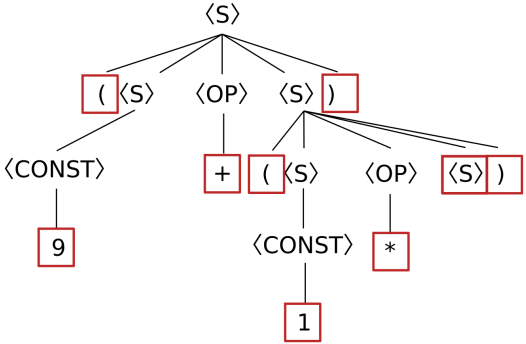
- Hashed path [Alon et al. 2019]
- Full tree
- Fringe

  Pattern:

  Fringe nodes

  Example tree

  $(9 + (1 * \langle S \rangle))$



# Equation view

**Input into discovery of one equation**

Dataset

| x | y |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

Unfinished syntax tree

S

Number Mul Product

Grammar:
S → Number Plus Sum
S → Number Mul Product
...
Variable → x

**Output of discovery of one equation**

Training data to improve

Complete syntax tree

S

Number Mul Product

1 * Variable

x

Reward for complete syntax tree

Reward
=
$1 - L_2(y_{dataset}, y_{prediction})$

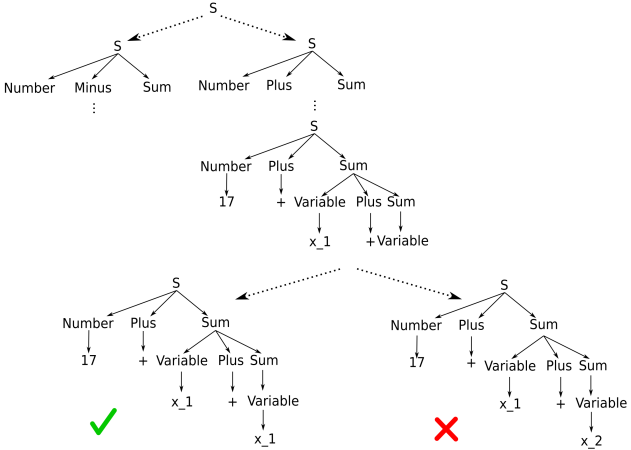
Intermediate syntax trees

S

Number Mul Product

MCTS visit counts

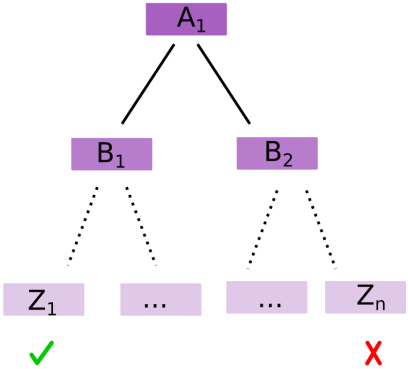
# Search tree


S
S
S
Number
Minus
Sum
Number
Plus
Sum
S
Number
Plus
Sum
17
+
Variable
Plus
Sum
x_1
+
Variable
S
Number
Plus
Sum
17
+
Variable
Plus
Sum
x_1
+
Variable
x_1
S
Number
Plus
Sum
17
+
Variable
Plus
Sum
x_1
+
Variable
x_2

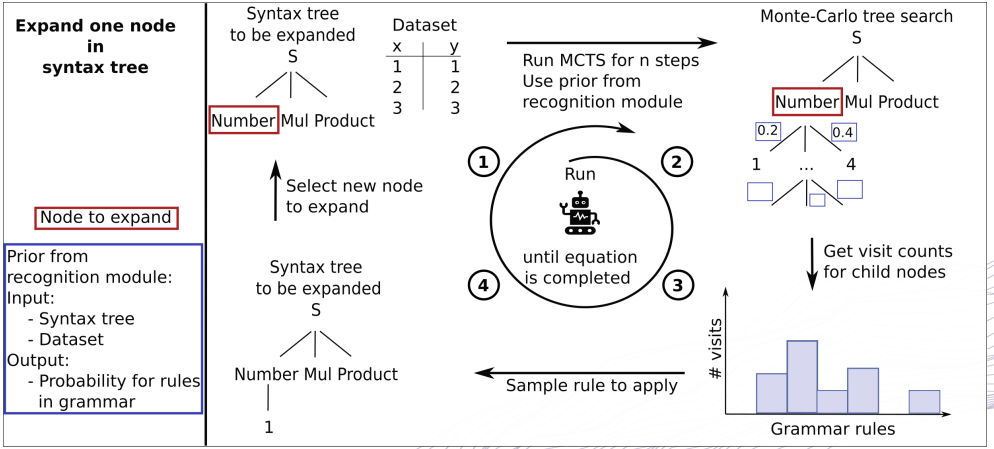
# Monte-Carlo tree search

Probability for child node depends on:

- Prior
  - Uniform
  - Grammar
  - Recognition module
- Visit counts
- Expected reward



# Recognition module



# Experiment: Sanity check

| Grammar |
| --- |
| *S* ::= *Number Plus S* |
| *S* ::= *Number* |
| *Number* ::= "1" |
| *Plus* ::= " + " |

Equations to discover

- $1$
- $1 + 1$
- ...
- $1 + 1 + 1 + 1 + 1$

True formula: $1$

"*S*" : $[0.09\ \underline{\mathbf{0.91}}]$

True formula: $1 + 1 + 1 + 1$

"*S*" : $[\underline{\mathbf{0.93}}\ 0.07]$

"$1 + S$" : $[\underline{\mathbf{0.84}}\ 0.16]$

"$1 + 1 + S$" : $[\underline{\mathbf{0.73}}\ 0.27]$

"$1 + 1 + 1 + S$" : $[0.46\ \underline{\mathbf{0.54}}]$

"$1 + 1 + 1 + 1 + S$" : $[0.38\ \mathbf{0.62}]$

# Experiment: Discovering equations

Grammar $[1/2]$:

- S $\rightarrow$ Number Plus Sum [0.2]
- S $\rightarrow$ Number Mul Product [0.15]
- S $\rightarrow$ Number Div OP Product CP [0.15]
- S $\rightarrow$ Number Minus OP Sum CP [0.1]
- S $\rightarrow$ Number Mul Trigonometric [0.1]
- S $\rightarrow$ Number Mul Logarithm [0.1]
- S $\rightarrow$ Number Mul Root [0.1]
- S $\rightarrow$ Number [0.1]
- Sum $\rightarrow$ Variable Plus Sum [0.3]
- Sum $\rightarrow$ Variable [0.7]
- Number $\rightarrow$ '1' [1.0]
- Number $\rightarrow$ '2' [0]
- Number $\rightarrow$ '3' [0]
- Number $\rightarrow$ '4' [0]
- Plus $\rightarrow$ '+' [1.0]
- Product $\rightarrow$ Variable Mul Product [0.3]
- Product $\rightarrow$ Variable [0.7]
- Mul $\rightarrow$ '*' [1.0]
- Div $\rightarrow$ '/' [1.0]
- Minus $\rightarrow$ '−' [1.0]

# Experiment: Discovering equations

Grammar $[2/2]$:

- Trigonometric $\rightarrow$ Trigonometric_operator OP Number Mul Variable CP [1.0]
- Trigonometric_operator $\rightarrow$ 'sin' [0.5]
- Trigonometric_operator $\rightarrow$ 'cos' [0.5]
- Logarithm $\rightarrow$ Logarithm_operator OP Number Mul Variable CP [1.0]
- Logarithm_operator $\rightarrow$ 'log' [1.0]
- Root $\rightarrow$ OP Number Plus Sum CP Power RootOperator [1.0]
- RootOperator $\rightarrow$ '0.5' [1.0]
- Power $\rightarrow$ '**' [1.0]
- OP $\rightarrow$ '(' [1.0]
- CP $\rightarrow$ ')' [1.0]  Variable $\rightarrow$ 'x' [1.0]

# Experiment: Discovering equations

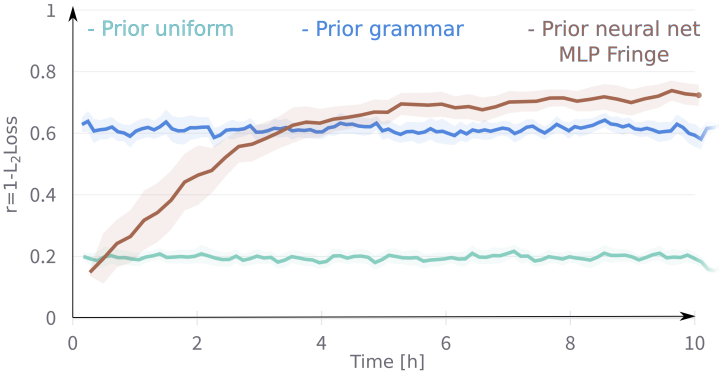
$x \sim \mathcal{N}(\mu = \{0.3, 2.24, 4.18, 6.12, 8.06, 10\}, \sigma^2 = 0.1)$

Equations:

- $y = \sqrt{1+x}$
- $y = \sqrt{1+x+x}$
- $y = \sqrt{1+x}$
- $y = cos(x)$
- $y = sin(x)$
- $y = log(x)$
- $y = x$
- $y = x + 1$
- $y = x^2$
- $y = x^3$
- $y = x + x + x$
- $y = x + x$
- $y = \frac{1}{x}$
- $y = \frac{1}{x^2}$
- $y = \frac{1}{x^3}$
- $y = 1 - x$
- $y = 1 - (x + x)$
- $y = 1 - (x + x + x)$
- $y = 1$

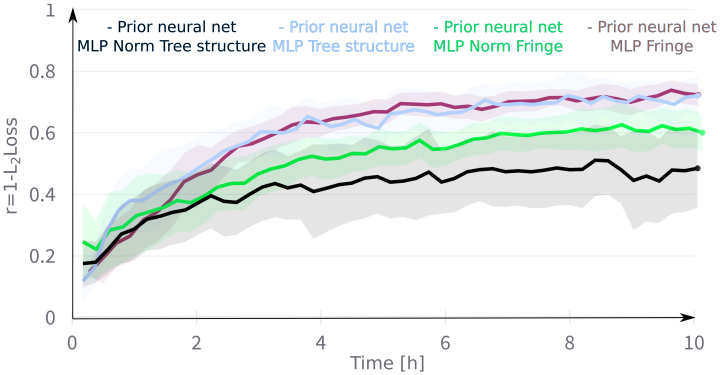
# Experiment: Discovering equations

100 MCTS steps



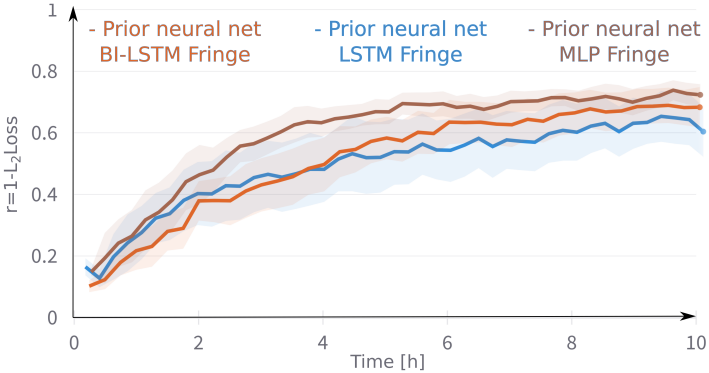
# Experiment: Discovering equations

100 MCTS steps



# Experiment: Discovering equations

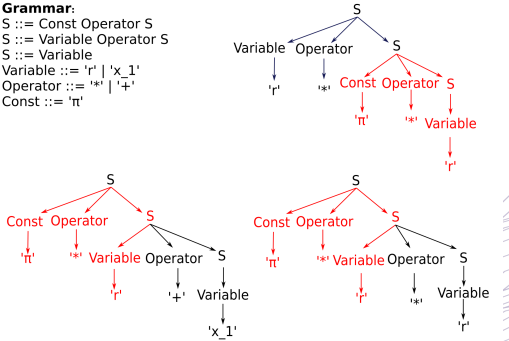
100 MCTS steps



# Outlook

- Dataset processing
- Find pattern in solution and add them to grammar

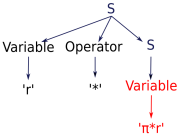
**Grammar**:
S ::= Const Operator S
S ::= Variable Operator S
S ::= Variable
Variable ::= 'r' | 'x_1'
Operator ::= '*' | '+'
Const ::= 'π'

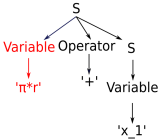


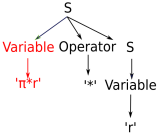
# Outlook

- Dataset processing
- Find pattern in solution and add them to grammar

**Grammar:**
S ::= Const Operator S
S ::= Variable Operator S
S ::= Variable
Variable ::= 'r' | 'x_1' |'π*r'
Operator ::= '*' | '+'
Const ::= 'π'



# Thank you for your attention

Conclusion:

- Symbolic regression
- Context-free grammar
- Representation of syntax tree
- System for neural-guided equation discovery
- Experiments

# Contact


- Jannis Brugger
- TU Darmstadt, Germany
- jannis.brugger@tu-darmstadt.de



- Mattia Cerrato
- JGU Mainz, Germany
- mcerrato@uni-mainz.de



- Stefan Kramer
- JGU Mainz, Germany
- kramer@informatik.uni-mainz.de

# Bibliography


- Alon, Uri et al. (2019). “Code2vec: Learning Distributed Representations of Code”. In: *Proc. ACM Program. Lang.* 3.POPL. DOI: 10.1145/3290353. URL: https://doi.org/10.1145/3290353.
- Borwein, J, P Borwein, and DH Bailey (1989). “Ramanujan, modular equations and approximations to pi”. In: *Amer. Math. Monthly* 96, pp. 201–219.
- Ellis, Kevin et al. (2021). “Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning”. In: *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pp. 835–850.
- Silver, David et al. (2018). “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419, pp. 1140–1144. DOI: 10.1126/science.aar6404. eprint: https://www.science.org/doi/pdf/10.1126/science.aar6404. URL: https://www.science.org/doi/abs/10.1126/science.aar6404.