

Deep Learning for **Symbolic Regression**

Anxiao (Andrew) Jiang

Computer Science and Engineering Department

Texas A&M University



“In 1601, Johannes Kepler got access to the world’s best data tables on planetary orbits.

And after 4 years and about 40 failed attempts to fit the Mars data to various ovoid shapes, he launched a scientific revolution by discovering that Mars’ orbit was an ellipse.

This was an example of **symbolic regression**: discovering a symbolic expression that accurately matches a given dataset.”

Can you see what function it is?

x_1	x_2	y
1	3	1.25
2	3	2.5
0	3	0
1	2	1.091
1	3.5	1.400
1.2	3.5	1.680
\vdots	\vdots	\vdots

Can you see what function it is?

x_1	x_2	y
1	3	1.25
2	3	2.5
0	3	0
1	2	1.091
1	3.5	1.400
1.2	3.5	1.680
\vdots	\vdots	\vdots

$$y = x_1 \cdot \frac{1}{\sqrt{1 - \frac{x_2^2}{25}}}$$

Can you see what function it is?

x_1	x_2	y
1	3	1.25
2	3	2.5
0	3	0
1	2	1.091
1	3.5	1.400
1.2	3.5	1.680
\vdots	\vdots	\vdots

$$y = x_1 \cdot \frac{1}{\sqrt{1 - \frac{x_2^2}{25}}}$$

Einstein's equation for
theory of special relativity

$$t' = t \cdot \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$$

Can you see what function it is?

x_1	x_2	x_3	x_3	y
1	1	1	1	50.265
1	1.5	2	1	87.965
1	1.5	3	8	640.885
1.05	1.5	3	8	672.929
1.05	1.5	3.1	8	694.041
1.1	1.6	3.1	3.8	369.904
\vdots	\vdots	\vdots	\vdots	\vdots

Can you see what function it is?

x_1	x_2	x_3	x_3	y
1	1	1	1	50.265
1	1.5	2	1	87.965
1	1.5	3	8	640.885
1.05	1.5	3	8	672.929
1.05	1.5	3.1	8	694.041
1.1	1.6	3.1	3.8	369.904
\vdots	\vdots	\vdots	\vdots	\vdots

$$y = 8\pi x_1(x_2 + x_3 x_4)$$

Can you see what function it is?

x_1	x_2	x_3	x_3	y
1	1	1	1	50.265
1	1.5	2	1	87.965
1	1.5	3	8	640.885
1.05	1.5	3	8	672.929
1.05	1.5	3.1	8	694.041
1.1	1.6	3.1	3.8	369.904
\vdots	\vdots	\vdots	\vdots	\vdots

$$y = 8\pi x_1(x_2 + x_3x_4)$$

Einstein's equation for
general theory of relativity

$$G_{\mu\nu} = 8\pi G(T_{\mu\nu} + \rho_{\Lambda}g_{\mu\nu})$$

Can you see what function it is?

x_1	x_2	x_3	y
1	1	2	4.014
1.5	1	2	1.590
1.5	2	3	5.575
1.5	4	3	4.447
1.5	4	3.6	5.846
2.8	4.2	3.6	5.182
\vdots	\vdots	\vdots	\vdots

Can you see what function it is?

x_1	x_2	x_3	y
1	1	2	4.014
1.5	1	2	1.590
1.5	2	3	5.575
1.5	4	3	4.447
1.5	4	3.6	5.846
2.8	4.2	3.6	5.182
\vdots	\vdots	\vdots	\vdots

$$y = \sqrt{\frac{4\pi^2}{9.8(x_1 + x_2)} \cdot x_3^3}$$

Can you see what function it is?

x_1	x_2	x_3	y
1	1	2	4.014
1.5	1	2	1.590
1.5	2	3	5.575
1.5	4	3	4.447
1.5	4	3.6	5.846
2.8	4.2	3.6	5.182
\vdots	\vdots	\vdots	\vdots

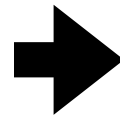
$$y = \sqrt{\frac{4\pi^2}{9.8(x_1 + x_2)}} \cdot x_3^3$$

Kepler's third law
on planetary movement

$$P^2 = \frac{4\pi^2}{G(m_1 + m_2)} \cdot a^3$$

Symbolic Regression

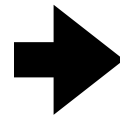
x_1	x_2	\dots	x_n	y
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots



$$y = f(x_1, x_2, \dots, x_n)$$

Symbolic Regression

x_1	x_2	\dots	x_n	y
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots

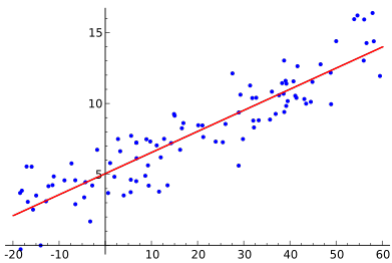


$$y = f(x_1, x_2, \dots, x_n)$$

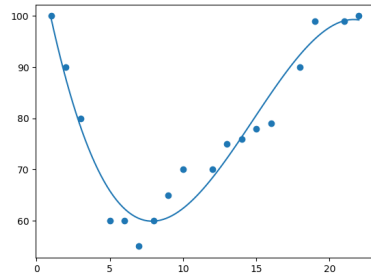
function f : Accurate & Simple

Symbolic Regression is different from

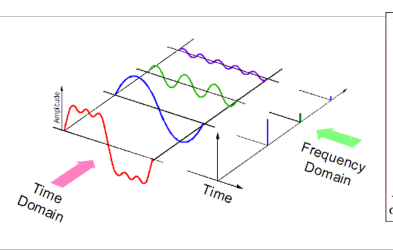
Linear regression



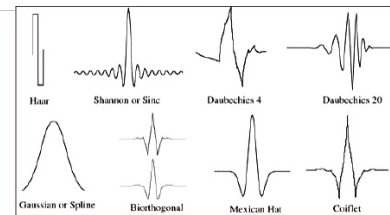
Polynomial regression



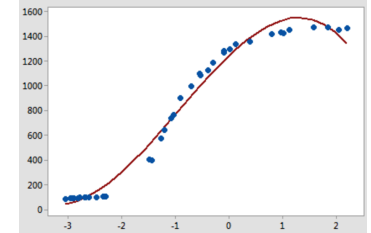
Fourier transform



Wavelet transform



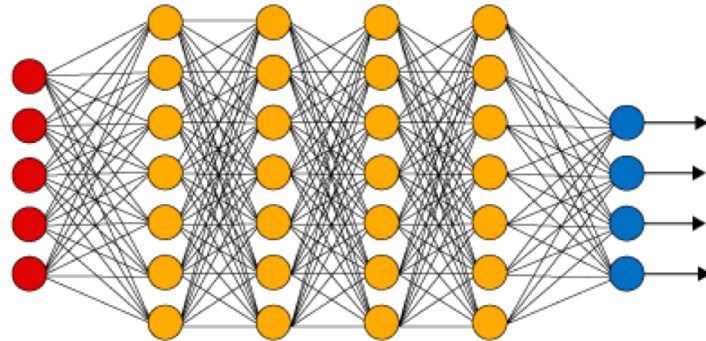
Nonlinear regression



because it needs to find both the **symbolic form** of the function as well as its **coefficients**.

Symbolic Regression is different from

Neural Networks



because it needs to find a simple form for the function.

Why is Symbolic Regression hard?

When the length of the function increases, the number of functions increases **exponentially**.

Consider functions of just ONE variable

Set of operations : $+$, \times , \sin , $\sqrt{\quad}$

Set of variables : x

Set of constants : 1 , -1

Consider functions of just ONE variable

Set of operations : $+$, \times , \sin , $\sqrt{\quad}$

Set of variables : x

Set of constants : 1 , -1

One operation: $x + 1$, $x - 1$, $-1 \times x$, $x \times x$, $\sin x$, \sqrt{x}

Consider functions of just ONE variable

Set of operations : $+$, \times , \sin , $\sqrt{\quad}$

Set of variables : x

Set of constants : 1 , -1

One operation: $x + 1$, $x - 1$, $-1 \times x$, $x \times x$, $\sin x$, \sqrt{x}

Two operations: $x + x + 1$, $x + x - 1$, $x \times x + 1$, $x \times x - 1$, $\sin x + 1$, $\sin x - 1$, $-\sin x$, $\sqrt{x} + 1$,

$\sqrt{x} - 1$, $-\sqrt{x}$, $\sin \sqrt{x}$, $\sqrt{\sin x}$, $\sin \sin x$, $\sqrt{\sqrt{x}}$...

Consider functions of just ONE variable

Set of operations : $+$, \times , \sin , $\sqrt{\quad}$

Set of variables : x

Set of constants : 1 , -1

When the length of the function increases, the number of functions increases exponentially.

One operation: $x + 1$, $x - 1$, $-1 \times x$, $x \times x$, $\sin x$, \sqrt{x}

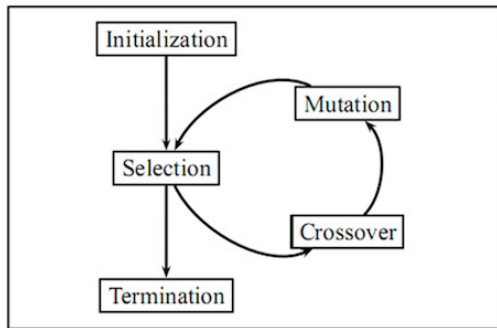
Two operations: $x + x + 1$, $x + x - 1$, $x \times x + 1$, $x \times x - 1$, $\sin x + 1$, $\sin x - 1$, $-\sin x$, $\sqrt{x} + 1$,

$\sqrt{x} - 1$, $-\sqrt{x}$, $\sin \sqrt{x}$, $\sqrt{\sin x}$, $\sin \sin x$, $\sqrt{\sqrt{x}}$...

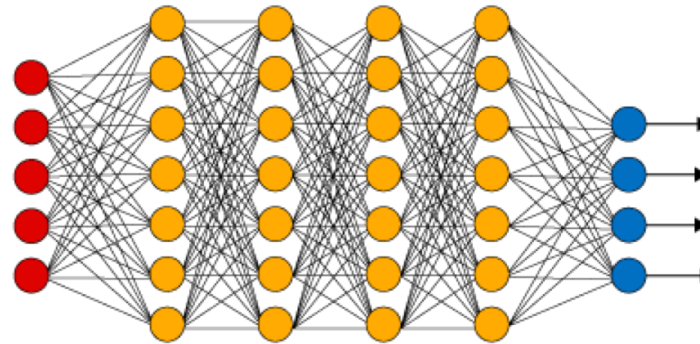
Three operations: $\sin(x) + x + 1$, $\sin x + x - 1$, $\sin(x) + \sin(x) + 1$, $\sin(x) + \sqrt{x} + 1$, $\sin(\sin(x)) + 1$,
 $\sqrt{\sqrt{x}} - 1$, $\sin(\sqrt{x}) + 1$, $\sqrt{\sin x} - 1$, $\sin x \times \sqrt{x}$, $\sin x - \sqrt{x}$, ...

Symbolic Regression Methods

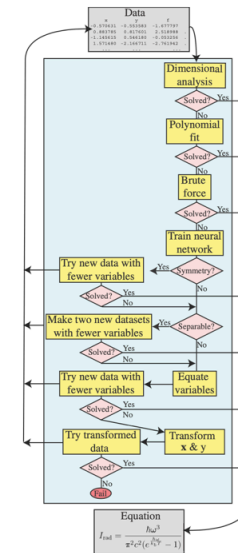
Evolutionary Algorithms



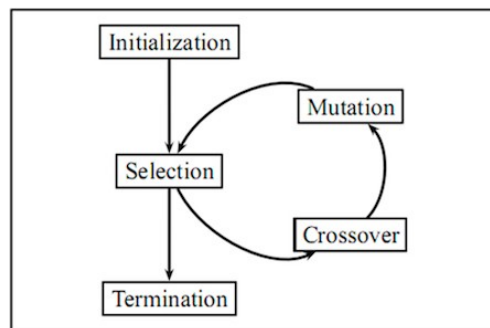
Neural Networks



Equation Simplification



Evolutionary Algorithms for Symbolic Regression



Evolutionary Algorithms for Symbolic Regression

1. Initial expressions are formed by randomly combining mathematical building blocks such as:

operations: $+$, $-$, \times , $/$, $\sqrt{\quad}$, \exp , \sin , \arcsin , etc.

variables: x_1, x_2, \dots, x_n

constants: $1, -1, 0, \pi, e, 1.2$, or a constant placeholder

Examples: $x_1 + 1, \sqrt{x_2}, x_1 + \sin(x_2), \sin(x_1) + x_2 - 1, \sqrt{\sin(x_1) + 1}, \frac{x_1}{x_2 + 1}, \frac{e^{x_1}}{x_2 + 1}, \dots$

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by **recombining previous equations** and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

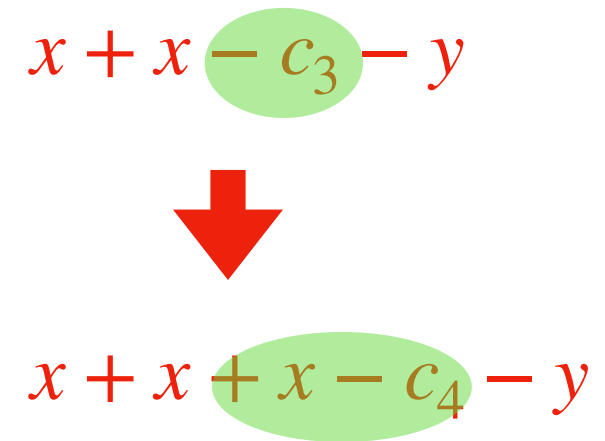



Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

$$x + x + x - c_4 - y$$


$$x + x + x - \sin(c_3) - y$$

Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by **recombining previous equations** and probabilistically varying their subexpressions.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

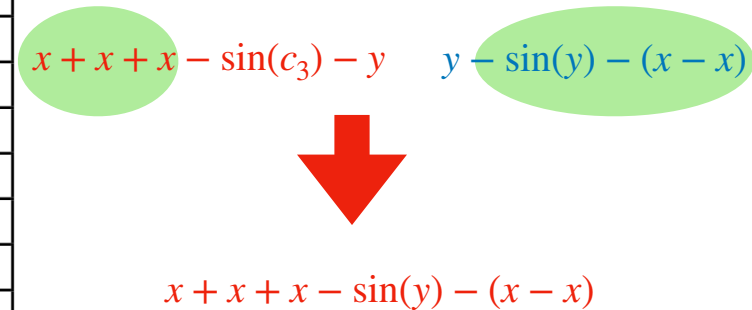


Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

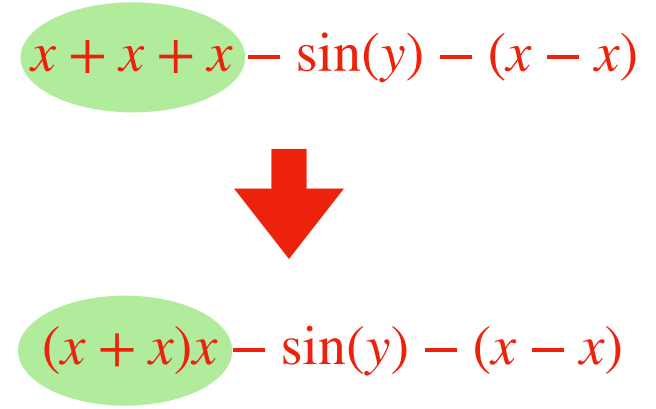


Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

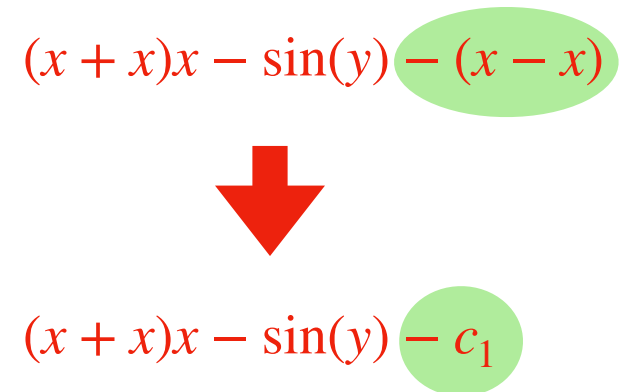


Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

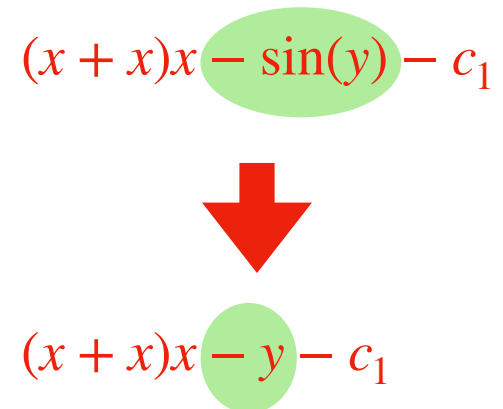


Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

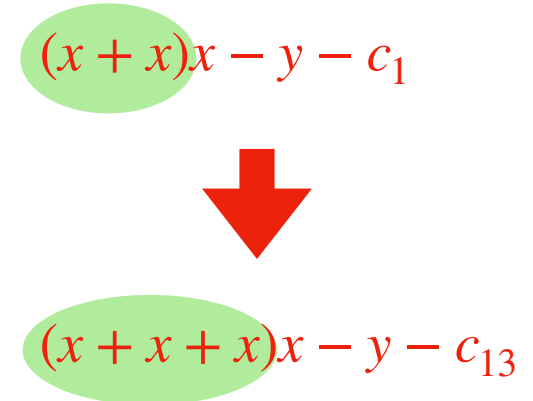


Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

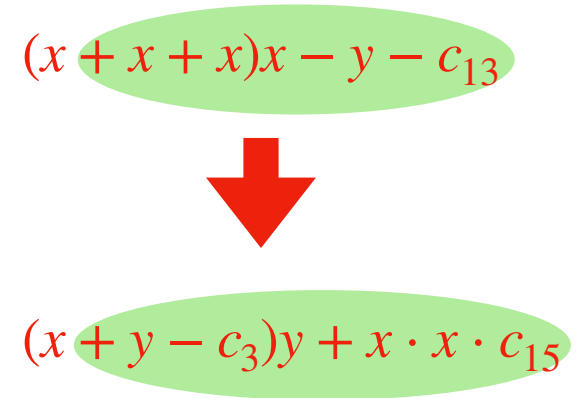



Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

$$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$$


$$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$$

Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by **recombining previous equations** and probabilistically varying their subexpressions.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

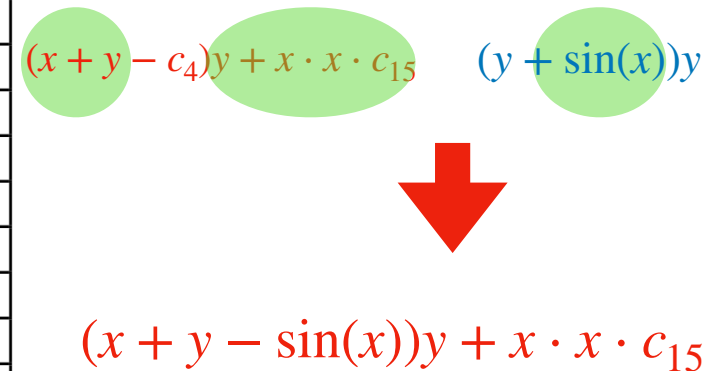


Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

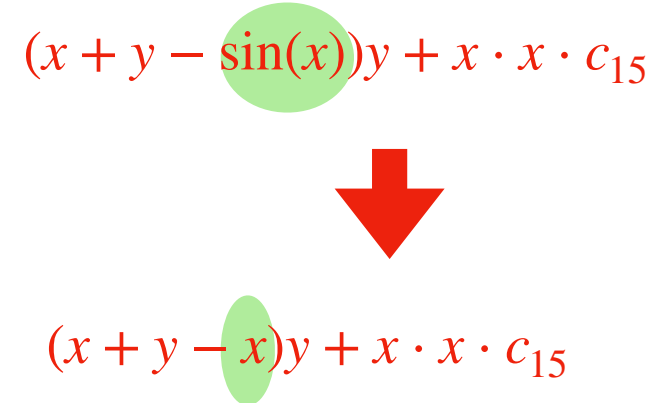



Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

$$(x + y - x)y + x \cdot x \cdot c_{15}$$


$$(x + y - x)y + x \cdot x \cdot c_{16}$$

Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

2. New equations are formed by recombining previous equations and **probabilistically varying their subexpressions**.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

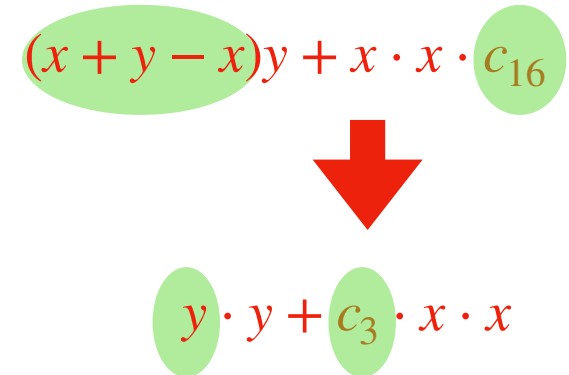


Fig. S2. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Fig. S1. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.

Evolutionary Algorithms for Symbolic Regression

3. The algorithm retains equations that model the experimental data better than others and abandons unpromising solutions.



Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

Evolutionary Algorithms for Symbolic Regression

4. After equations reach a desired level of accuracy, the algorithm terminates, returning a set of equations that are most likely to correspond to the intrinsic mechanisms underlying the observed system.

Accuracy	Equations in Sequence	Event
-1.4197	$x + x - c_3 - y$	<i>random</i>
-1.41347	$x + x + x - c_4 - y$	<i>mutation</i>
-1.41339	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
-1.13805	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
-1.08904	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
-1.08574	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
-1.01841	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
-0.978484	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
-0.914336	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.303559	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0692607	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
-0.0140815	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
-0.0050732	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
-0.0050732	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

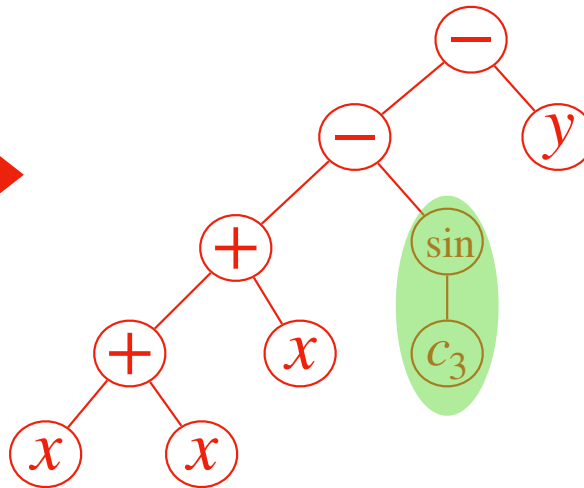
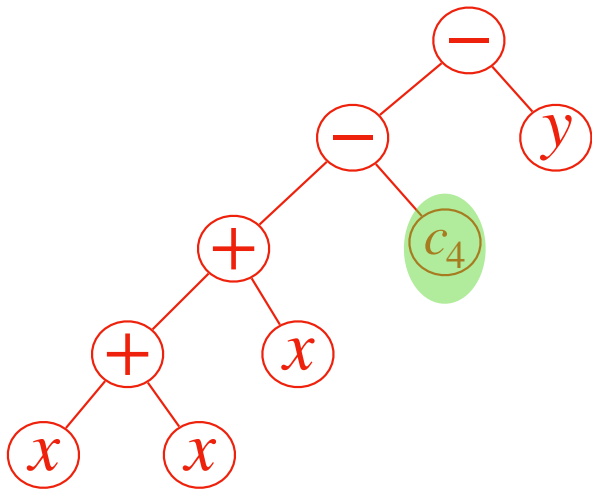
Two Basic Steps in an Evolutionary Algorithm

Mutation

$$x + x + x - c_4 - y$$



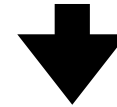
$$x + x + x - \sin(c_3) - y$$



Crossover

$$x + x + x - \sin(c_3) - y$$

$$y - \sin(y) - (x - x)$$



$$x + x + x - \sin(y) - (x - x)$$

Two Basic Steps in an Evolutionary Algorithm

Mutation

$$x + x + x - c_4 - y$$



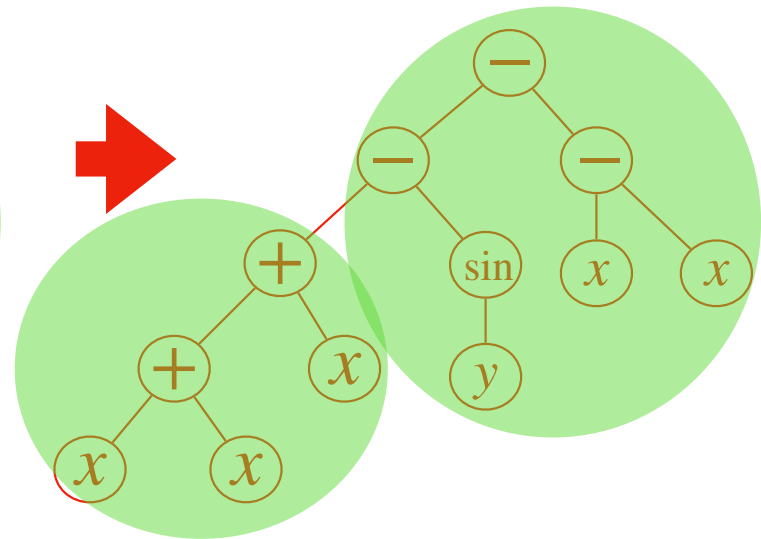
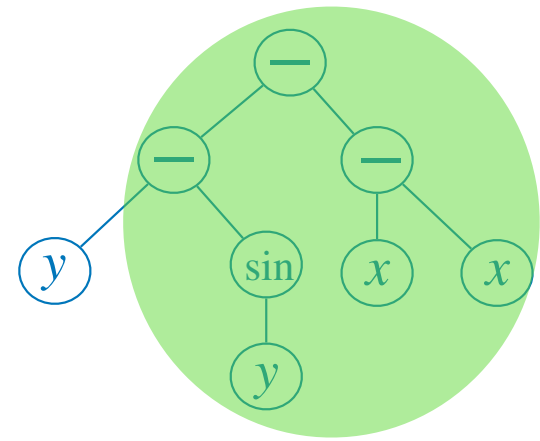
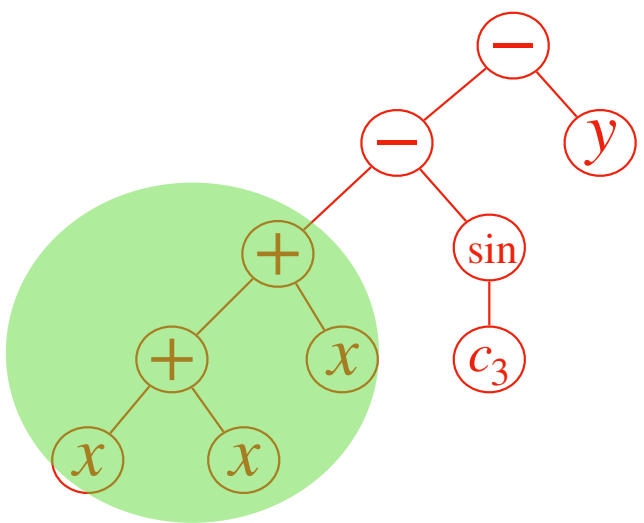
$$x + x + x - \sin(c_3) - y$$

Crossover

$$x + x + x - \sin(c_3) - y \quad y - \sin(y) - (x - x)$$



$$x + x + x - \sin(y) - (x - x)$$



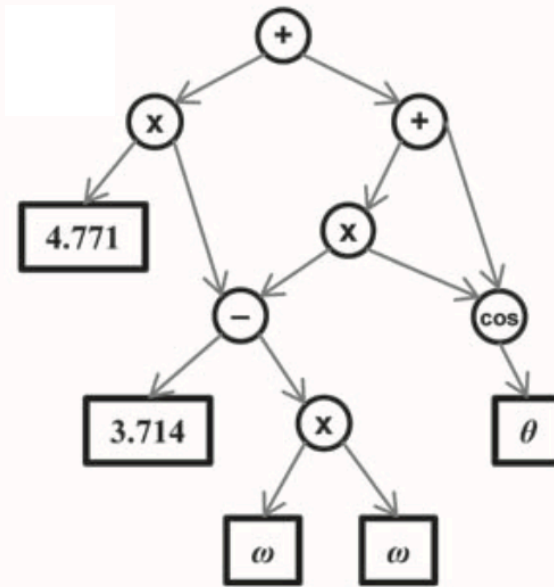
Evolutionary Algorithms for Symbolic Regression: How to use prior knowledge

Seed the equation search by initializing the algorithm's initial set of candidate equations with terms from equations from simpler systems (that were previously studied).

Computation Tree can also be represented by a
Directed Acyclic Computation Graph

```

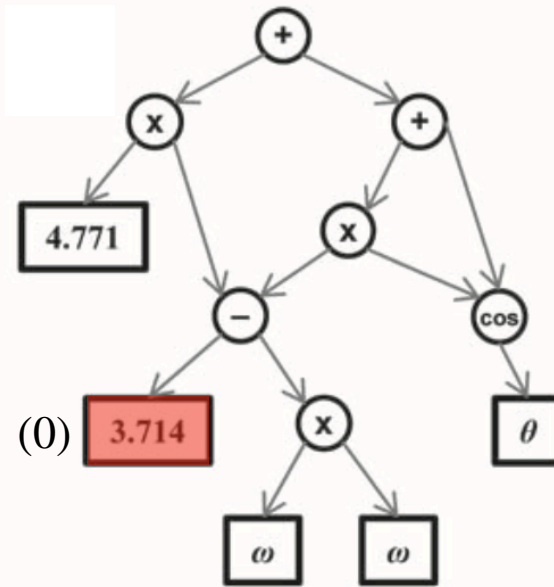
$$f(\theta, \omega) = 4.771 \cdot (3.714 - \omega^2) + \cos(\theta) + (3.714 - \omega^2) \cdot \cos(\theta)$$
  
(0) <- load [3.714]  
(1) <- load [ $\omega$ ]  
(2) <- mul (1), (1)  
(3) <- sub (0), (2)  
(4) <- load [ $\theta$ ]  
(5) <- cos (4)  
(6) <- mul (3), (5)  
(7) <- load [4.771]  
(8) <- mul (7), (3)  
(9) <- add (6), (5)  
(10) <- add (9), (8)
```



Computation Tree can also be represented by a
Directed Acyclic Computation Graph

```

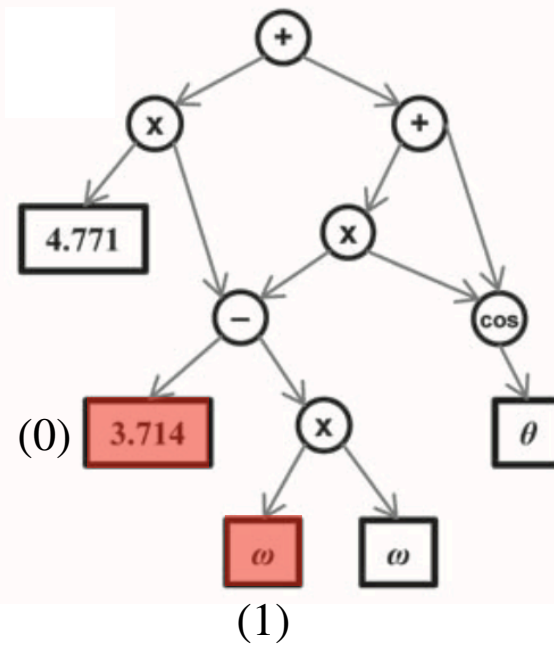
$$f(\theta, \omega) = 4.771 \cdot (3.714 - \omega^2) + \cos(\theta) + (3.714 - \omega^2) \cdot \cos(\theta)$$
  
(0) <- load [3.714]  
(1) <- load [ $\omega$ ]  
(2) <- mul (1), (1)  
(3) <- sub (0), (2)  
(4) <- load [ $\theta$ ]  
(5) <- cos (4)  
(6) <- mul (3), (5)  
(7) <- load [4.771]  
(8) <- mul (7), (3)  
(9) <- add (6), (5)  
(10) <- add (9), (8)
```



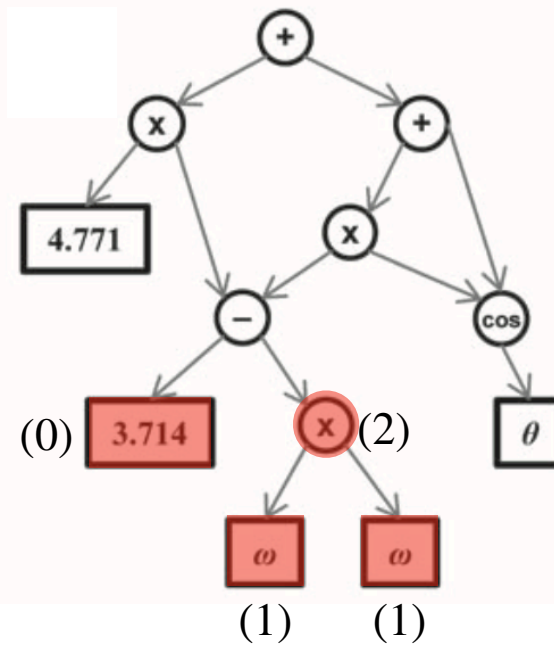
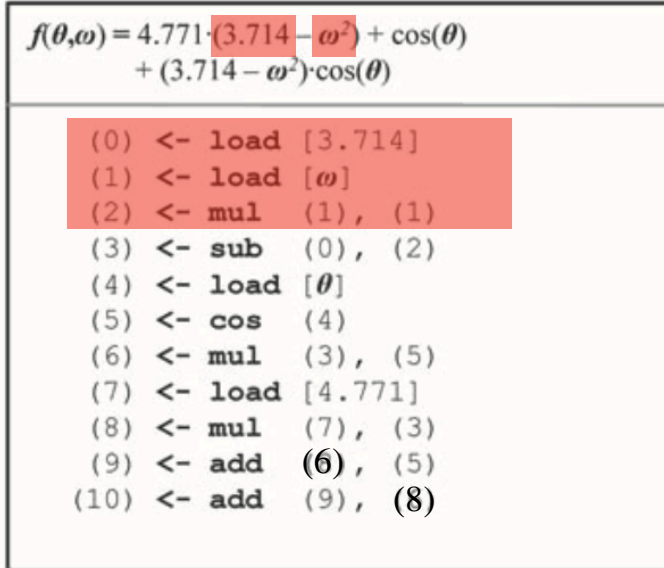
Computation Tree can also be represented by a
Directed Acyclic Computation Graph

```

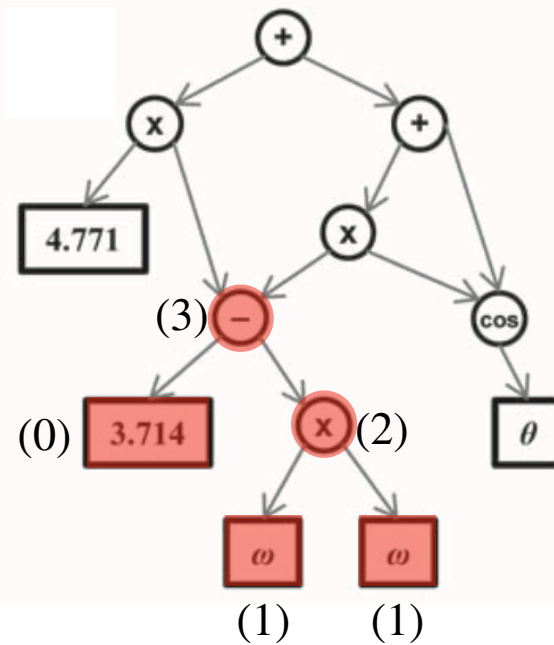
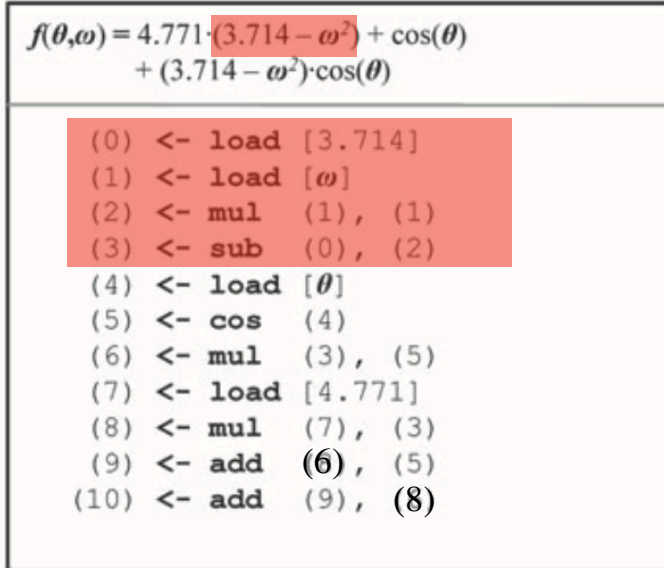
$$f(\theta, \omega) = 4.771 \cdot (3.714 - \omega^2) + \cos(\theta) + (3.714 - \omega^2) \cdot \cos(\theta)$$
  
(0) <- load [3.714]  
(1) <- load [ $\omega$ ]  
(2) <- mul (1), (1)  
(3) <- sub (0), (2)  
(4) <- load [ $\theta$ ]  
(5) <- cos (4)  
(6) <- mul (3), (5)  
(7) <- load [4.771]  
(8) <- mul (7), (3)  
(9) <- add (6), (5)  
(10) <- add (9), (8)
```



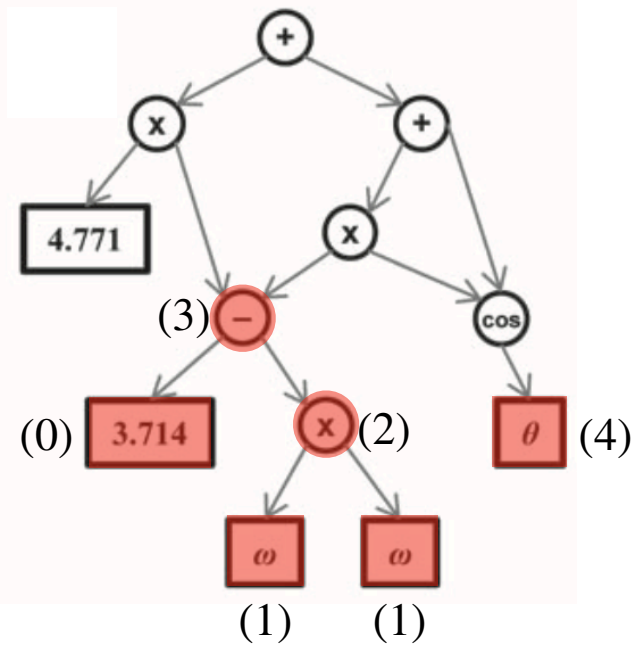
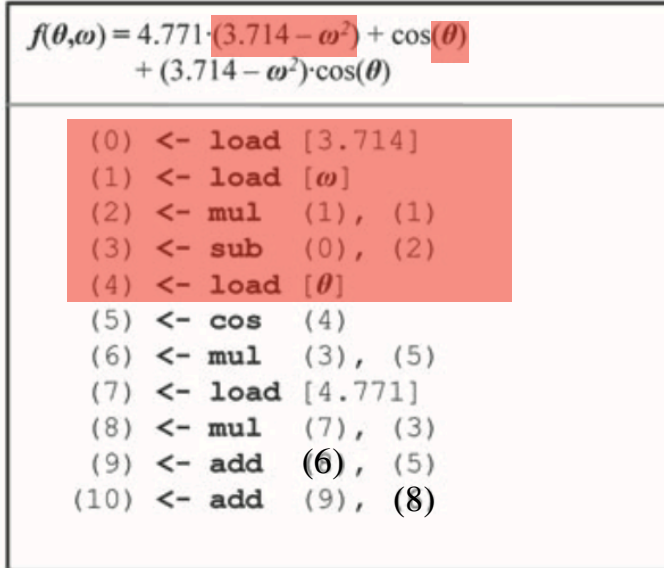
Computation Tree can also be represented by a
Directed Acyclic Computation Graph



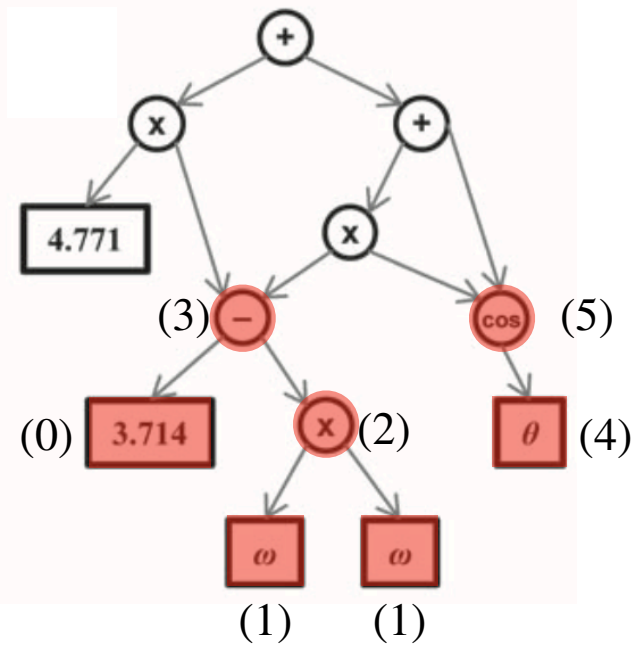
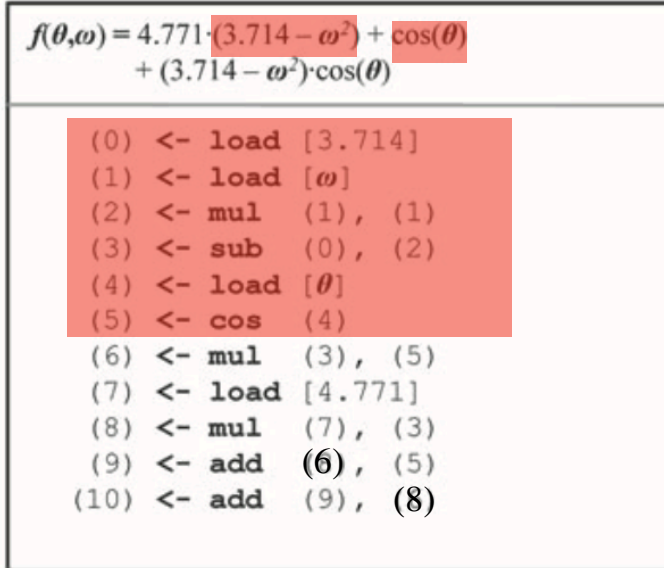
Computation Tree can also be represented by a
Directed Acyclic Computation Graph



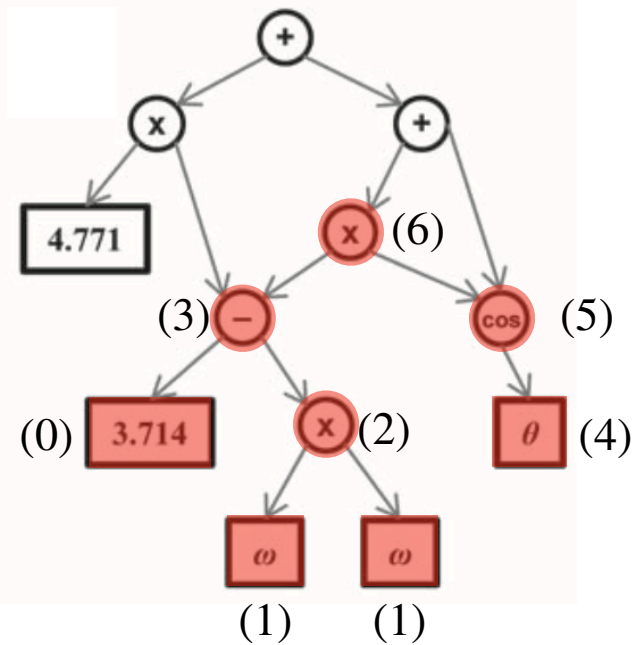
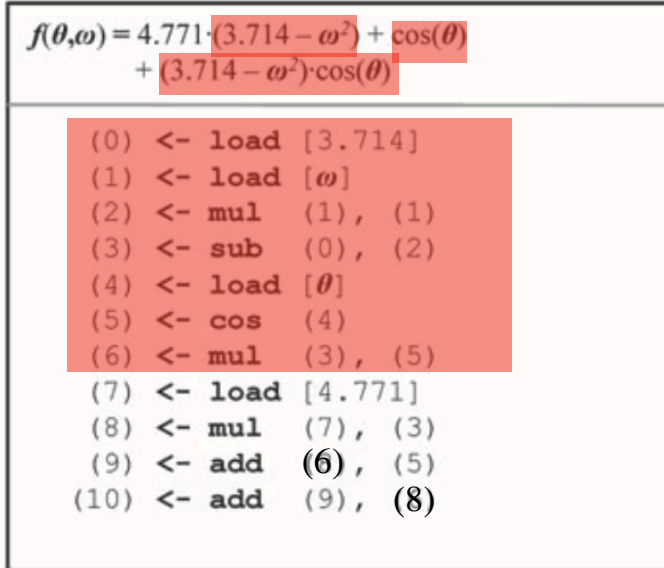
Computation Tree can also be represented by a
Directed Acyclic Computation Graph



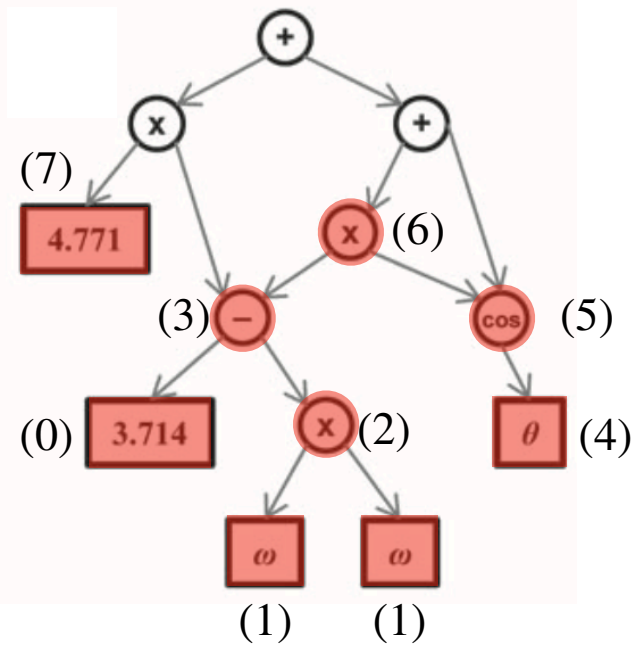
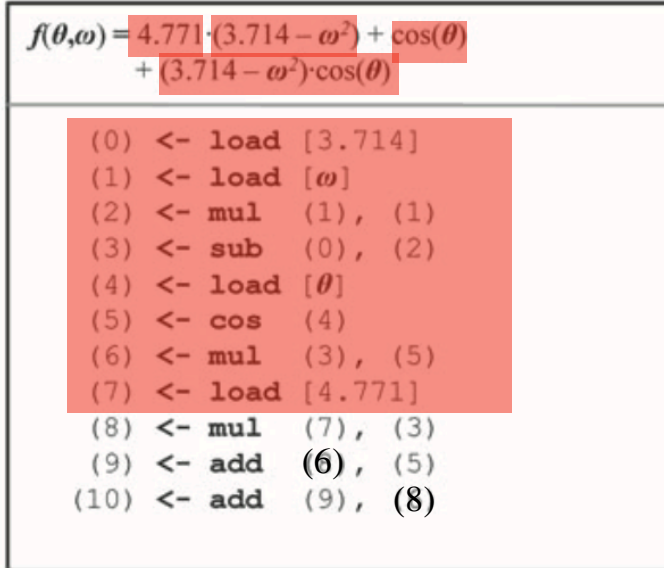
Computation Tree can also be represented by a
Directed Acyclic Computation Graph



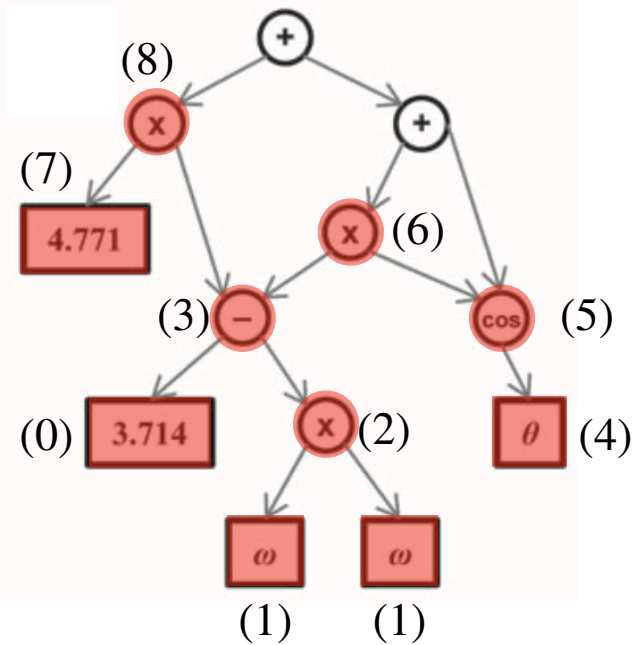
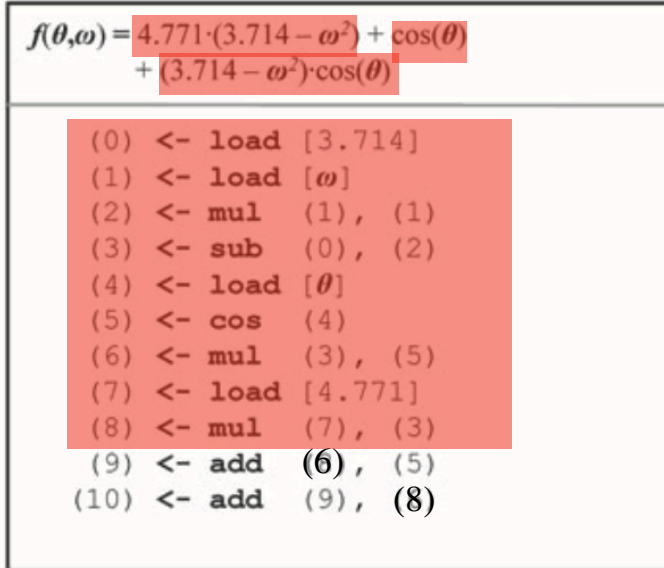
Computation Tree can also be represented by a
Directed Acyclic Computation Graph



Computation Tree can also be represented by a
Directed Acyclic Computation Graph



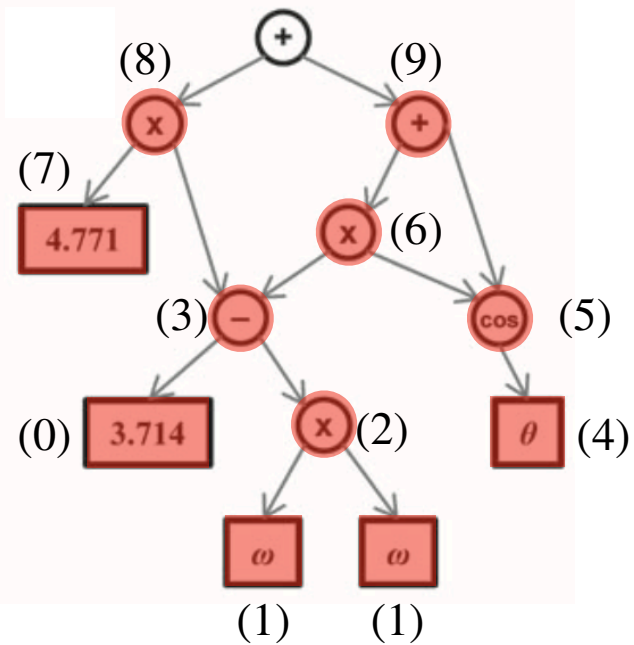
Computation Tree can also be represented by a
Directed Acyclic Computation Graph



Computation Tree can also be represented by a
Directed Acyclic Computation Graph

```
 $f(\theta, \omega) = 4.771 \cdot (3.714 - \omega^2) + \cos(\theta)$   
 $+ (3.714 - \omega^2) \cdot \cos(\theta)$ 
```

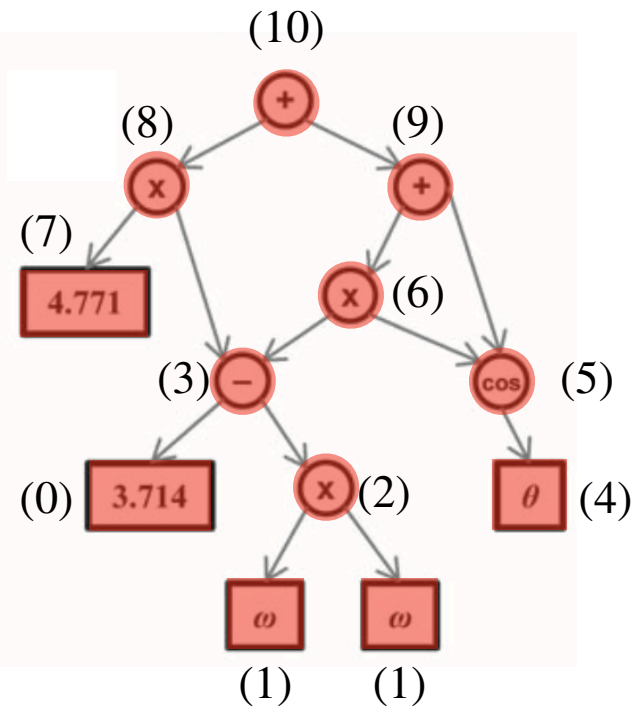
```
(0) <- load [3.714]  
(1) <- load [ $\omega$ ]  
(2) <- mul (1), (1)  
(3) <- sub (0), (2)  
(4) <- load [ $\theta$ ]  
(5) <- cos (4)  
(6) <- mul (3), (5)  
(7) <- load [4.771]  
(8) <- mul (7), (3)  
(9) <- add (6), (5)  
(10) <- add (9), (8)
```



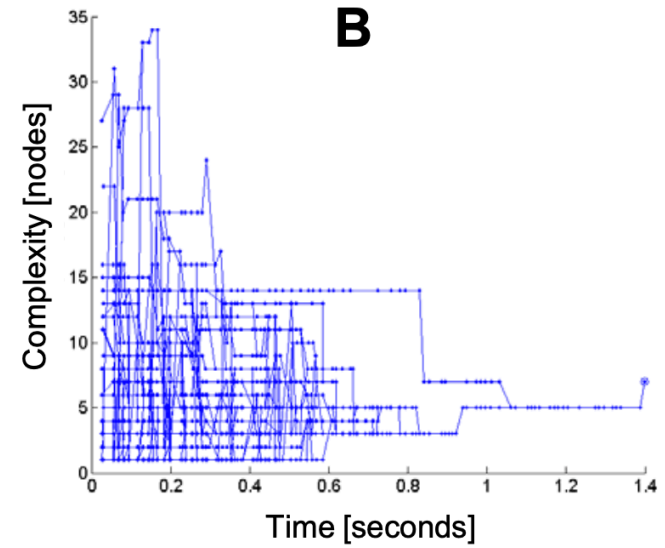
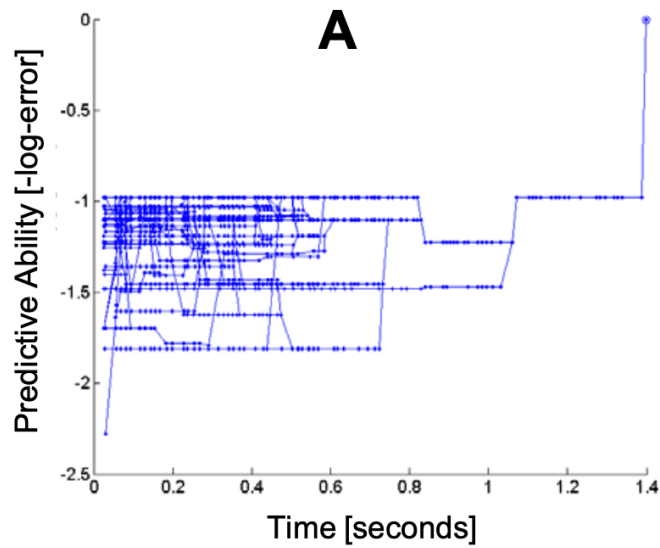
Computation Tree can also be represented by a
Directed Acyclic Computation Graph

```
 $f(\theta, \omega) = 4.771 \cdot (3.714 - \omega^2) + \cos(\theta) + (3.714 - \omega^2) \cdot \cos(\theta)$ 
```

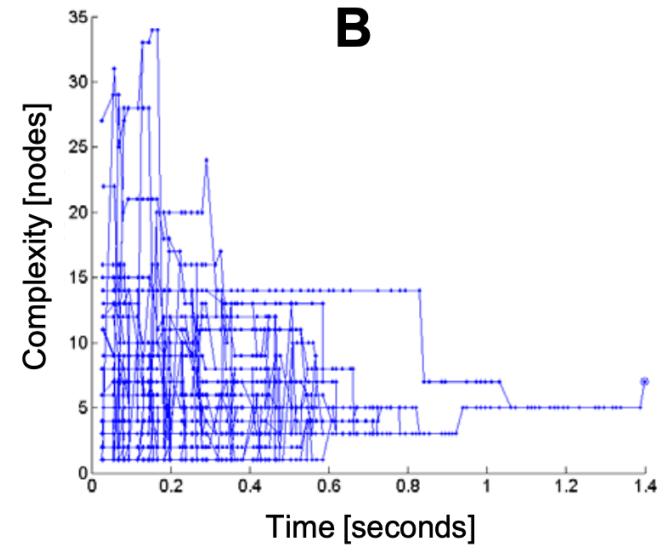
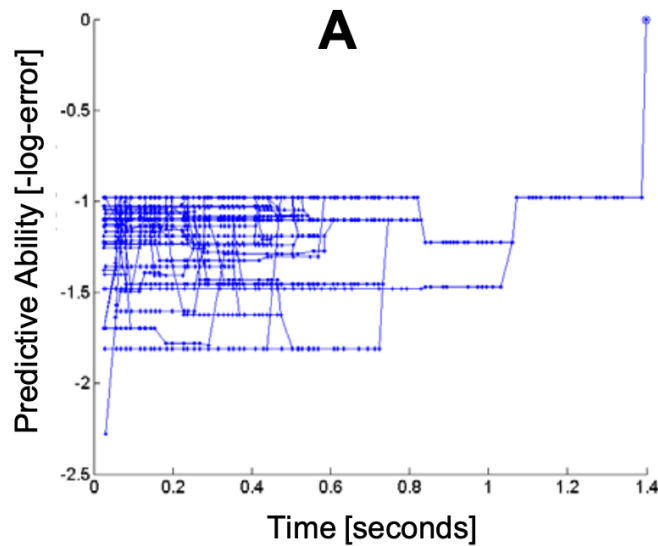
```
(0) <- load [3.714]  
(1) <- load [ $\omega$ ]  
(2) <- mul (1), (1)  
(3) <- sub (0), (2)  
(4) <- load [ $\theta$ ]  
(5) <- cos (4)  
(6) <- mul (3), (5)  
(7) <- load [4.771]  
(8) <- mul (7), (3)  
(9) <- add (6), (5)  
(10) <- add (9), (8)
```



Accuracy & Complexity of Found Equations (evolution over time)



Accuracy & Complexity of Found Equations (evolution over time)



- 1) Computation can take a long time for complex equations.
- 2) It is much more time consuming to find symbolic forms of equations than to compute their constant coefficients.

Use Symbolic Regression to find Laws in Physics

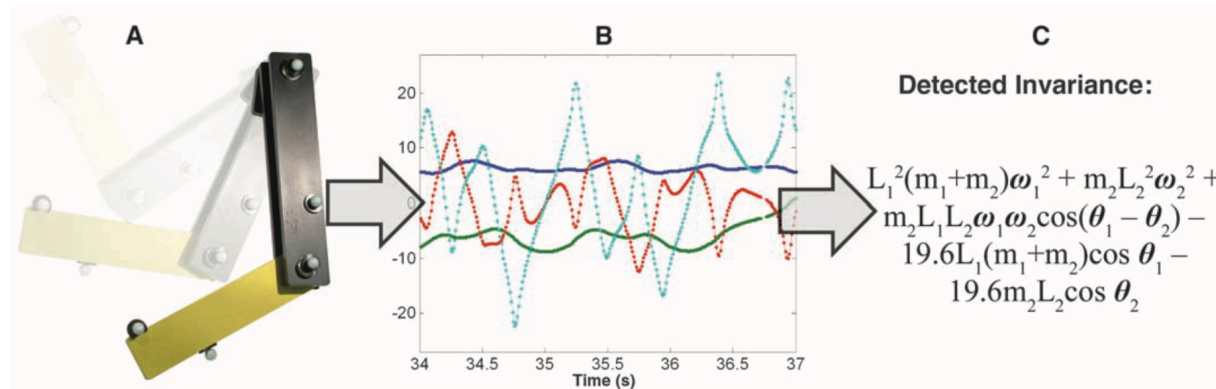
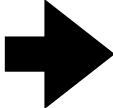


Fig. 1. Mining physical systems. We captured the angles and angular velocities of a chaotic double-pendulum (A) over time using motion tracking (B), then we automatically searched for equations that describe a single natural law relating

these variables. Without any prior knowledge about physics or geometry, the algorithm found the conservation law (C), which turns out to be the double pendulum's Hamiltonian. Actual pendulum, data, and results are shown.

Symbolic Regression can find equations
(more general relations than functions)

Function		Equation
$y = f(x_1, x_2, \dots, x_n)$		$y - f(x_1, x_2, \dots, x_n) = 0$

More general equation: $F(x_1, x_2, \dots, x_n, y) = C$

Example: $x^2 + y^2 = 1$

Avoid finding trivial relations, such as

$$\sin^2(3.1x) + \cos^2(3.1x) = 1$$

$$\sin^2(3.2x) + \cos^2(3.2x) = 1$$

$$x_1 + 4.56 - x_2 x_1 / x_2 = 4.56$$

Idea: use derivatives to find non-trivial relations

We define a potential equation to be nontrivial if it can predict differential relationships between two or more variables.

Use partial derivatives between pairs of variables

Consider equation $f(x, y) = C$, where $x = x(t)$ and $y = y(t)$ are both time-series data.

Then $\frac{\partial f / \partial y}{\partial f / \partial x} = \frac{\partial x}{\partial y} \approx \frac{dx/dt}{dy/dt} \approx \frac{\Delta x}{\Delta y}$.

compute using function f measure using experimental data

We can now compare $\Delta x / \Delta y$ values from the experimental data with $\partial x / \partial y$.

Compute values from the candidate function $f(x, y)$ to see how well they match:

$$-\frac{1}{N} \sum_{i=1}^N \log \left(1 + \left| \frac{\Delta x_i}{\Delta y_i} - \frac{\partial x_i}{\partial y_i} \right| \right)$$

Instead of squared-error, mean error, correlation, etc., we can use the mean-log-error for numerical reasons (which is robust against outliers).

Use partial derivatives between pairs of variables

Consider equation $f(x, y) = C$, where $x = x(t)$ and $y = y(t)$ are both time-series data.

Then $\frac{\partial f / \partial y}{\partial f / \partial x} = \frac{\partial x}{\partial y} \approx \frac{dx/dt}{dy/dt} \approx \frac{\Delta x}{\Delta y}$.

compute using function f measure using experimental data

We can now compare $\Delta x / \Delta y$ values from the experimental data with $\partial x / \partial y$.

Compute values from the candidate function $f(x, y)$ to see how well they match:

$$-\frac{1}{N} \sum_{i=1}^N \log \left(1 + \left| \frac{\Delta x_i}{\Delta y_i} - \frac{\partial x_i}{\partial y_i} \right| \right)$$

Instead of squared-error, mean error, correlation, etc., we can use the mean-log-error for numerical reasons (which is robust against outliers).

Use partial derivatives between pairs of variables

Consider equation $f(x, y) = C$, where $x = x(t)$ and $y = y(t)$ are both time-series data.

Then $\frac{\partial f / \partial y}{\partial f / \partial x} = \frac{\partial x}{\partial y} \approx \frac{dx/dt}{dy/dt} \approx \frac{\Delta x}{\Delta y}$.

compute using function f measure using experimental data

We can now compare $\Delta x / \Delta y$ values from the experimental data with $\partial x / \partial y$.

Compute values from the candidate function $f(x, y)$ to see how well they match:

$$-\frac{1}{N} \sum_{i=1}^N \log \left(1 + \left| \frac{\Delta x_i}{\Delta y_i} - \frac{\partial x_i}{\partial y_i} \right| \right)$$

Instead of squared-error, mean error, correlation, etc., we can use the mean-log-error for numerical reasons (which is robust against outliers).

Use partial derivatives between pairs of variables

Consider equation $f(x, y) = C$, where $x = x(t)$ and $y = y(t)$ are both time-series data.

Then
$$\frac{\partial f / \partial y}{\partial f / \partial x} = \frac{\partial x}{\partial y} \approx \frac{dx/dt}{dy/dt} \approx \frac{\Delta x}{\Delta y}.$$

compute using function f measure using experimental data

We can now compare $\Delta x / \Delta y$ values from the experimental data with $\partial x / \partial y$.

Compute values from the candidate function $f(x, y)$ to see how well they match:

$$-\frac{1}{N} \sum_{i=1}^N \log \left(1 + \left| \frac{\Delta x_i}{\Delta y_i} - \frac{\partial x_i}{\partial y_i} \right| \right)$$

Instead of squared-error, mean error, correlation, etc., we can use the mean-log-error for numerical reasons (which is robust against outliers).

Find multiple equations using Pareto-front

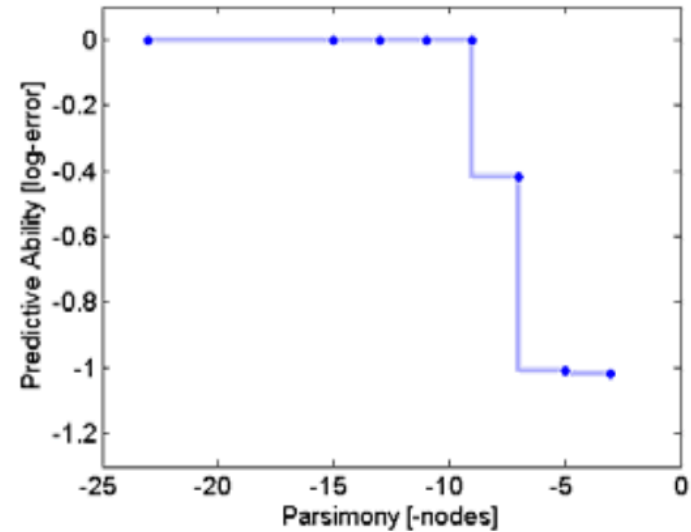
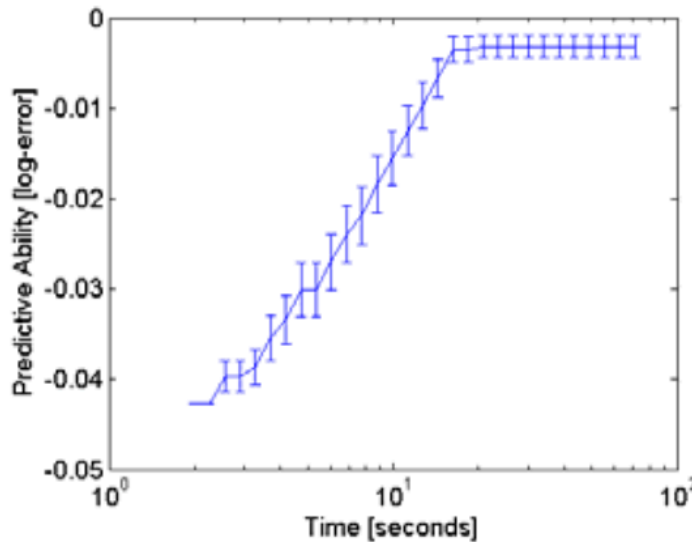
Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Circle:
 $x^2 + y^2$



Find multiple equations using Pareto-front

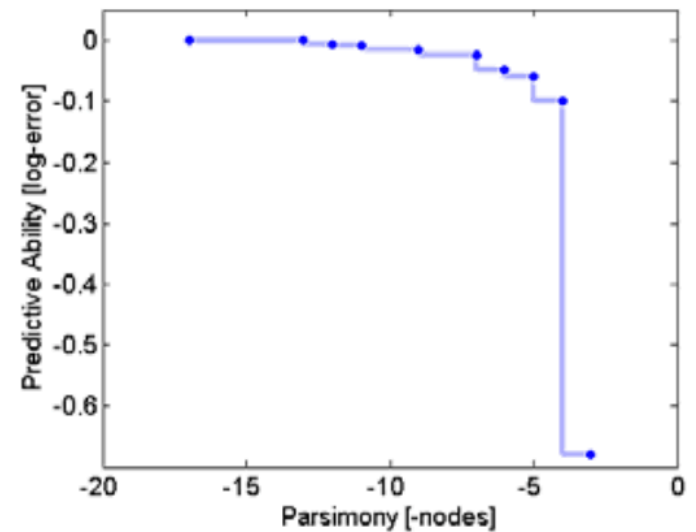
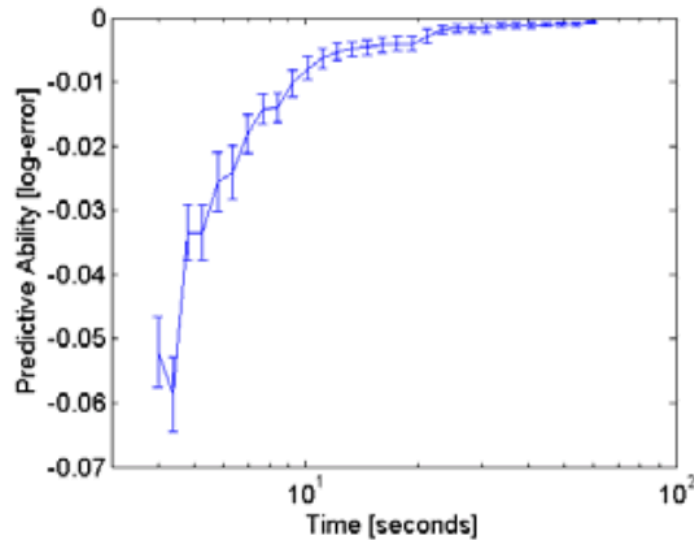
Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Elliptic Curve:
 $x^3 + x - y^2$



Find multiple equations using Pareto-front

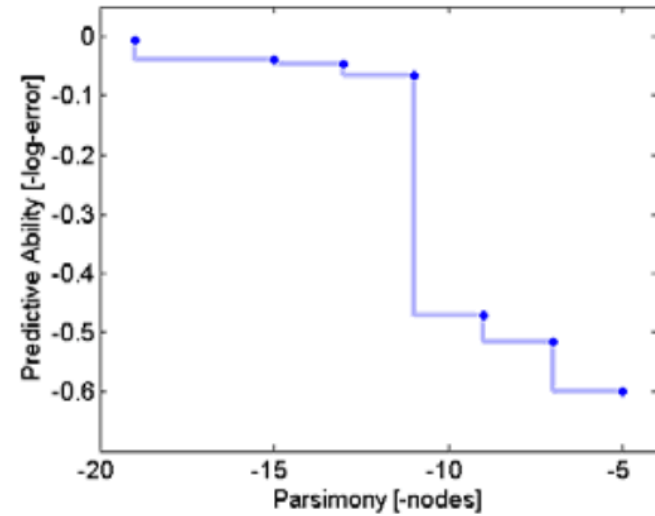
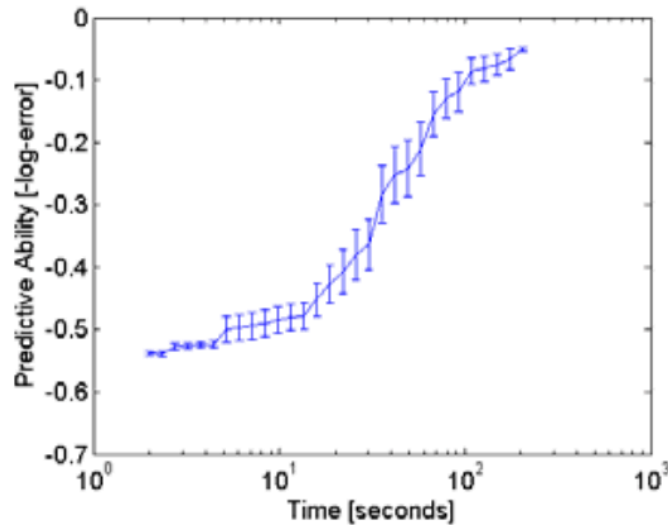
Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Sphere:
 $x^2 + y^2 + z^2$



Find multiple equations using Pareto-front

Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

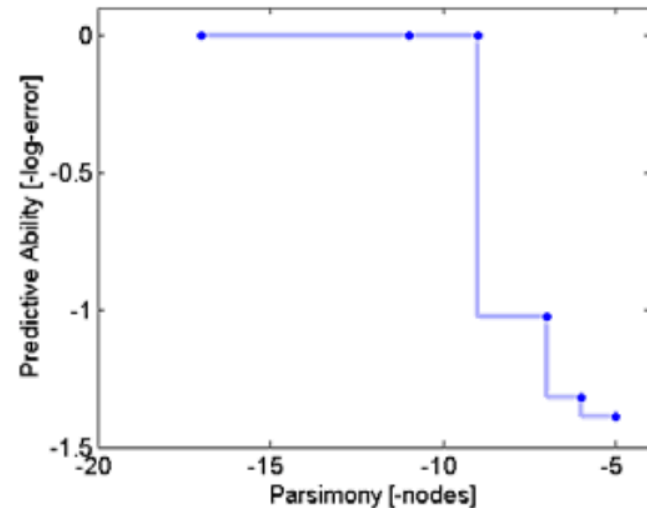
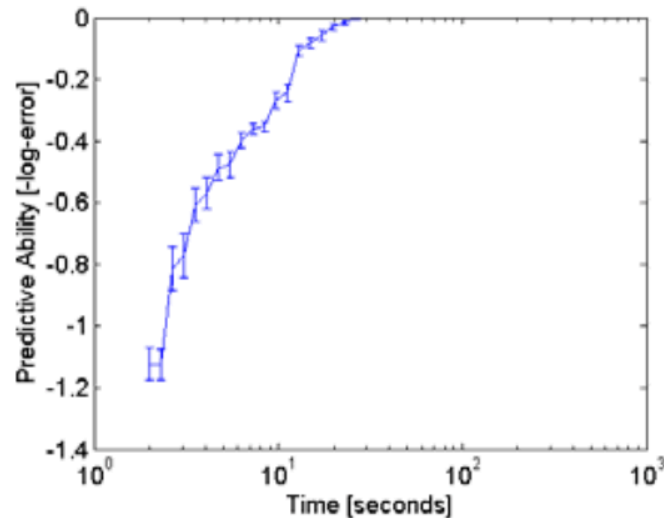
System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Linear Oscillator:

$$a - 0.1 \cdot v + 3 \cdot x$$



Find multiple equations using Pareto-front

Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

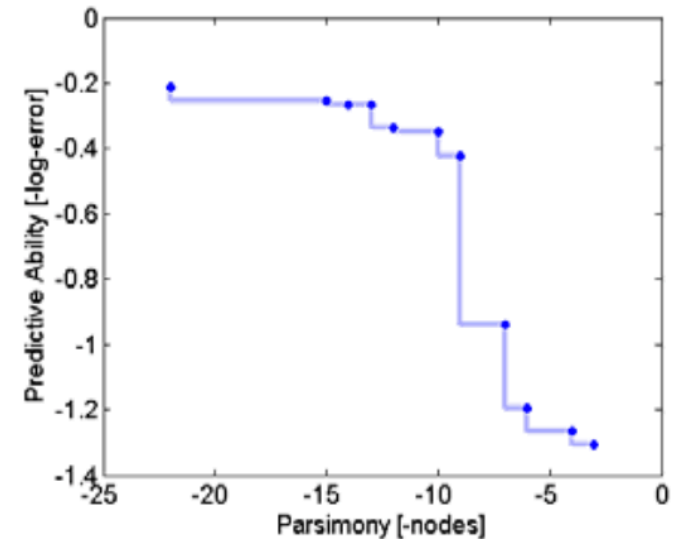
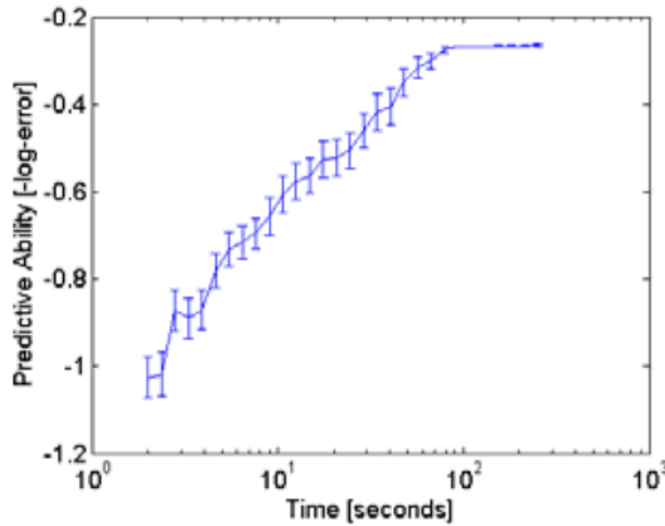
System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Linear Oscillator:

$$x^2 + 0.3 \cdot v^2$$



Find multiple equations using Pareto-front

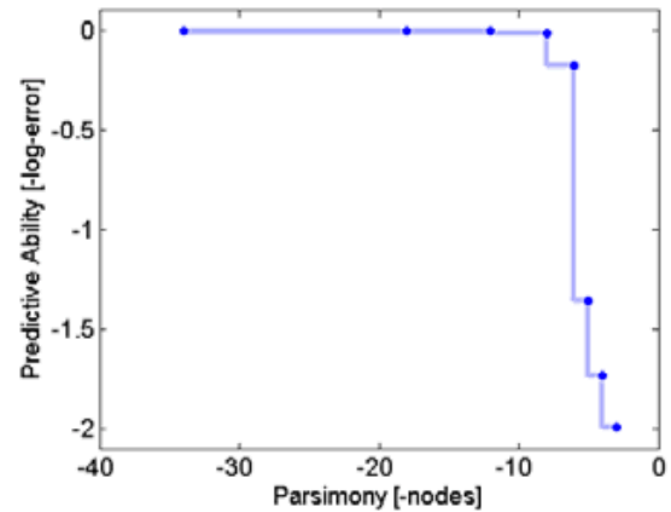
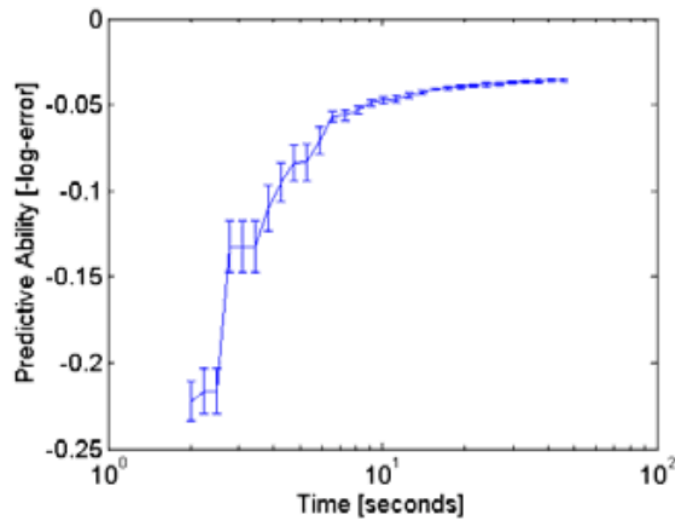
Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Pendulum:
 $\alpha - 9.8 \cdot \sin(\theta)$



Find multiple equations using Pareto-front

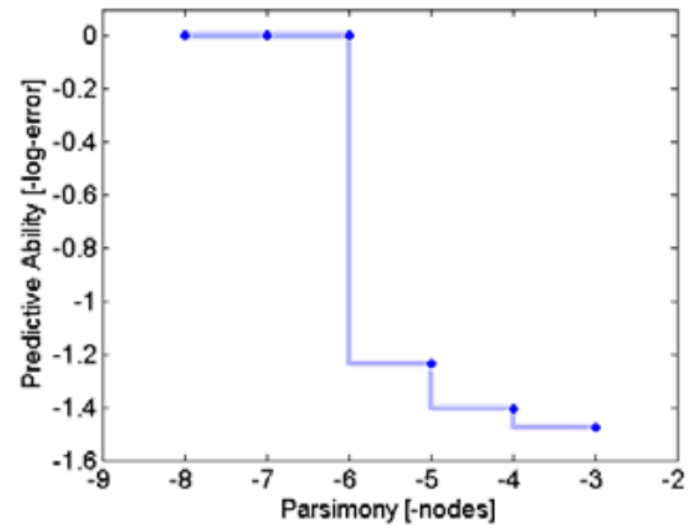
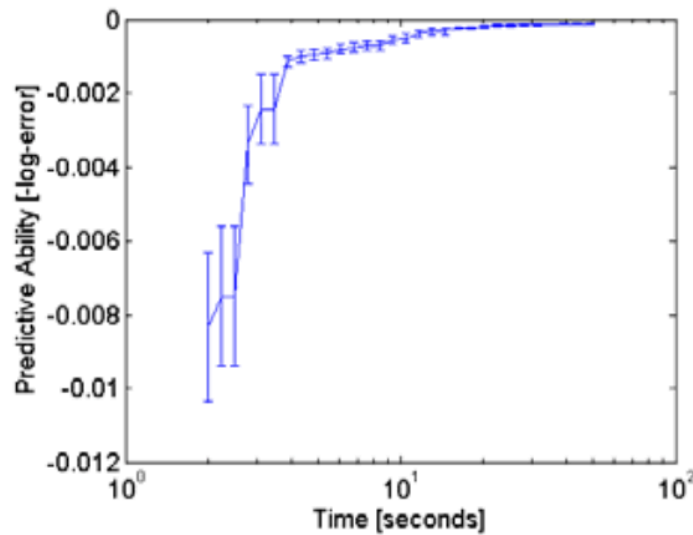
Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Pendulum:
 $\omega^2 - 9.8 \cdot \cos(\theta)$



Find multiple equations using Pareto-front

Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

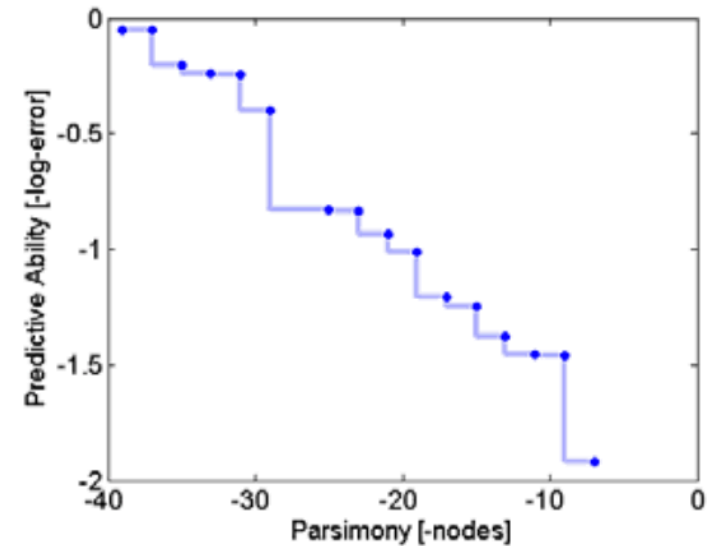
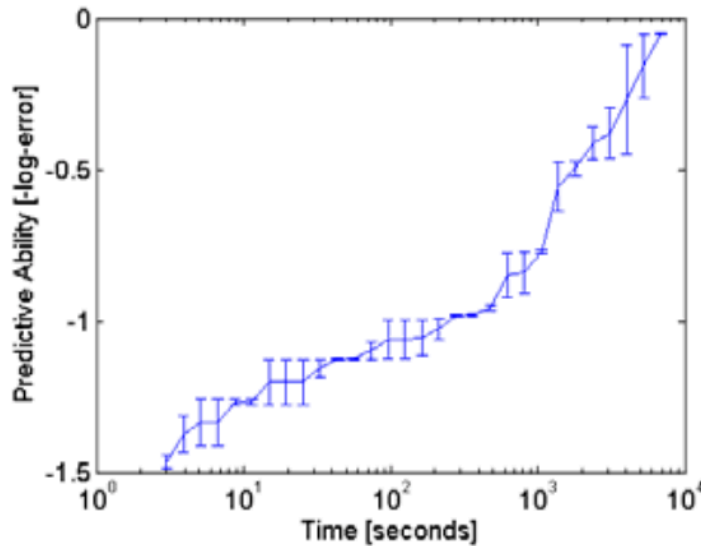
System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

**Double Linear
Oscillator**

$$x_1^2 + (x_1 - x_2)^2 + (1 - x_2)^2 + 2 \cdot v_1^2 + v_2^2$$



Find multiple equations using Pareto-front

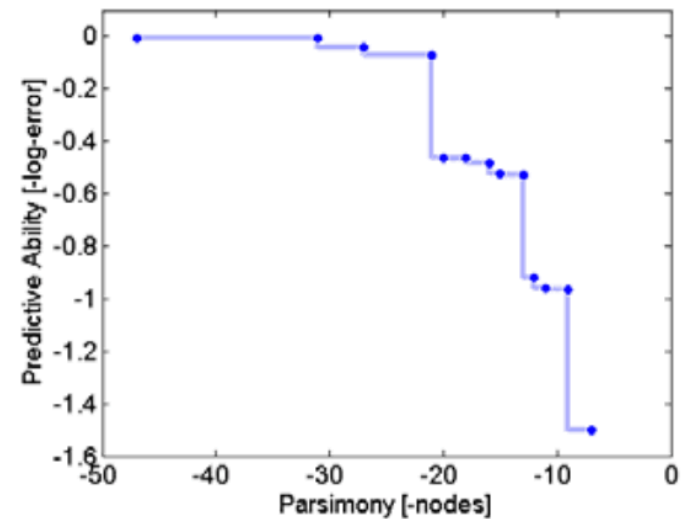
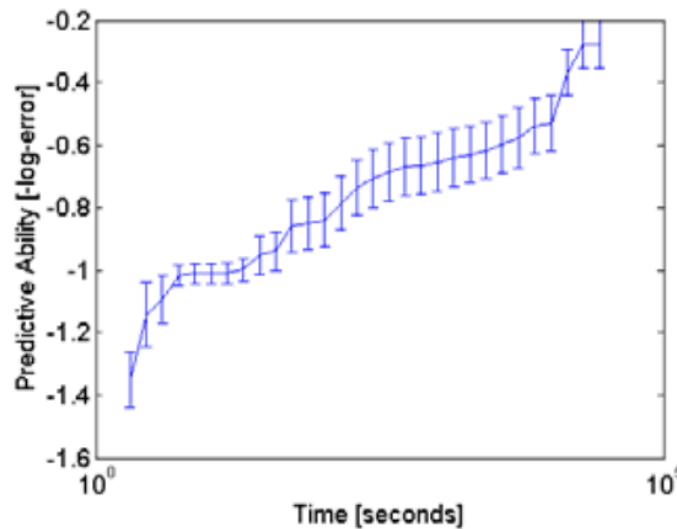
Pareto-front: optimal tradeoff between **accuracy** & **complexity** of the equation.

System

Predictive Ability Over Time

Accuracy/Complexity Pareto Front

Double Pendulum
 $\omega_1^2 + 0.5 \cdot \omega_2^2 +$
 $\omega_1 \omega_2 \cos(\theta_1 - \theta_2) -$
 $19.6 \cos(\theta_1) - 9.8 \cos(\theta_2)$



Impact of noise

Noise can make symbolic regression significantly more difficult. In particular, noise makes approximating the gradient (numerical derivatives) more difficult because derivatives can be highly sensitive to noise.

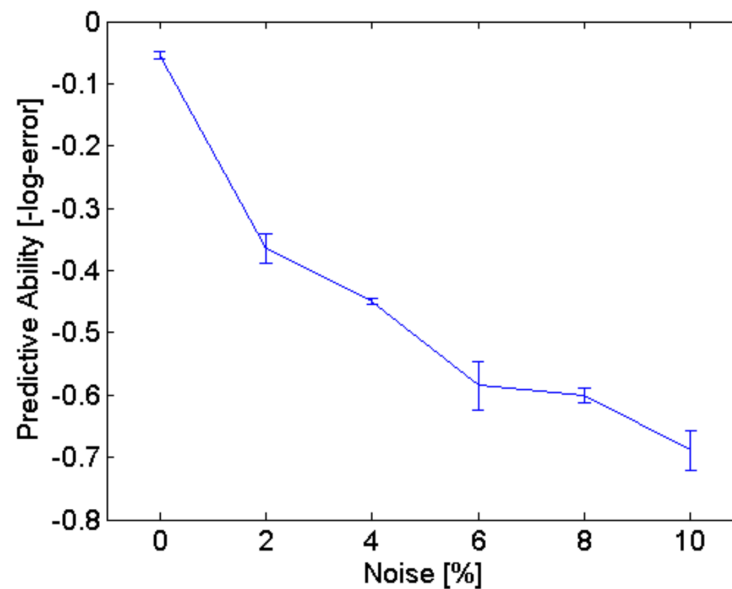


Fig. S5. The mean predictive ability on a withheld test set of the best law equations detected versus the amount of normally distributed noise in the data set for the simulated double linear oscillator. Error bars show the standard error. The percent noise is the ratio of the standard deviation of the noise and the standard deviation of the original signal.

Impact of noise

We can use Loess smoothing - a non-parametric fitting method - to remove high frequency noise from data. Loess smoothing updates each sample in the dataset by fitting a small order polynomial to the sample and its nearest neighbors.

Other methods, such as filtering and convolution, also reduce high-frequency noise, but do not readily produce estimates of the signal derivative. Using Loess smoothing, we can obtain the numerical derivatives directly from the smoothing procedure by evaluating the symbolic derivatives of the local polynomial fits at each data sample.

We can measure the noise strength (percent noise) as the ratio of the standard deviation of the random noise to the standard deviation of the exact signal.

Impact of noise

We can use Loess smoothing - a non-parametric fitting method - to remove high frequency noise from data. Loess smoothing updates each sample in the dataset by fitting a small order polynomial to the sample and its nearest neighbors.

Other methods, such as filtering and convolution, also reduce high-frequency noise, but do not readily produce estimates of the signal derivative. Using Loess smoothing, we can obtain the numerical derivatives directly from the smoothing procedure by evaluating the symbolic derivatives of the local polynomial fits at each data sample.

We can measure the noise strength (percent noise) as the ratio of the standard deviation of the random noise to the standard deviation of the exact signal.

Impact of noise

We can use Loess smoothing - a non-parametric fitting method - to remove high frequency noise from data. Loess smoothing updates each sample in the dataset by fitting a small order polynomial to the sample and its nearest neighbors.

Other methods, such as filtering and convolution, also reduce high-frequency noise, but do not readily produce estimates of the signal derivative. Using Loess smoothing, we can obtain the numerical derivatives directly from the smoothing procedure by evaluating the symbolic derivatives of the local polynomial fits at each data sample.

We can measure the noise strength (percent noise) as the ratio of the standard deviation of the random noise to the standard deviation of the exact signal.

Build Up an alphabet for symbolic regression (for future experiments)

Method: extract common subtrees of good equations that are found via symbolic regression.

Term	Complexity	Frequency	Systems
$k x_2$	1	23.578	4
$k x_1$	1	21.6514	4
$k v_2$	1	21.5596	4
$k v_1$	1	19.1743	4
$k \cos(\theta_2)$	2	6.51376	2
$k \cos(\theta_1)$	2	6.14679	2
$k v_2^2$	3	5.50459	4
$k v_1^2$	3	4.49541	4
$(x_1 - x_2)$	3	4.22018	2
$k v_1 v_2$	3	4.0367	2
$k a_2$	1	2.75229	2
$k a_1$	1	2.75229	2
$k x_2^2$	3	1.37615	2
$k \sin(\theta_2)$	2	1.19266	2
$k \sin(\theta_1)$	2	1.19266	2
$k \cos(\theta_1)$	2	0.917431	2
$k \cos(\theta_2)$	2	0.917431	2
$k x_1^2$	3	0.825688	2
$k x_2^2$	3	0.825688	2
$k x_1 + k v_1$	3	0.733945	3
$k x_2 + k v_2$	3	0.733945	3
$k x_1 + k x_2$	3	0.642202	2
$k x_1 + k v_1 - k a_1$	5	0.458716	2
$k x_2 + k v_2 - k a_2$	5	0.458716	2

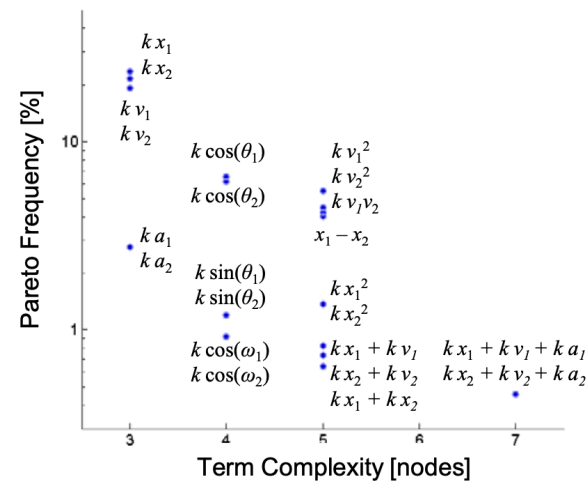
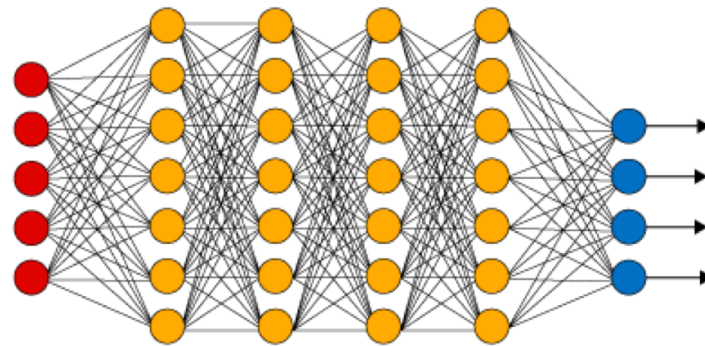


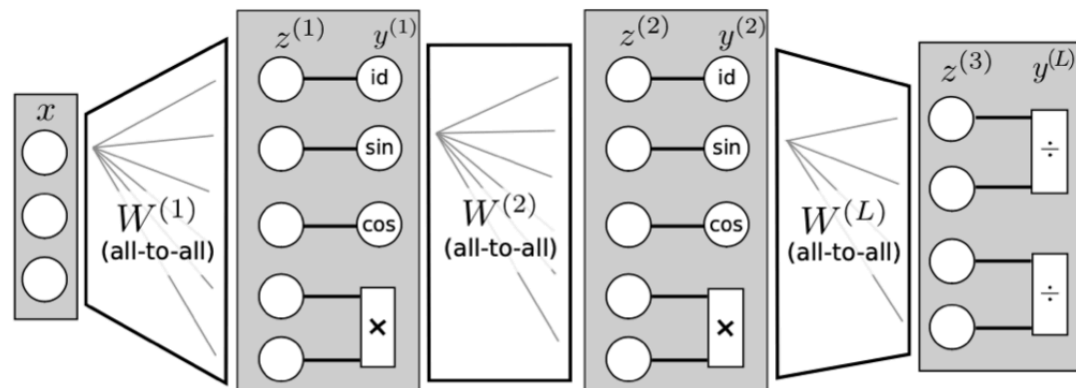
Fig. S6. The occurrence of common terms among the pareto fronts of the linear oscillator, double linear oscillator, pendulum, and double pendulum sorted by frequency of appearance. Several terms re-emerge between these systems revealing a common physical language for kinetic and potential energy, trigonometry, sum of forces.

Neural Network for Symbolic Regression



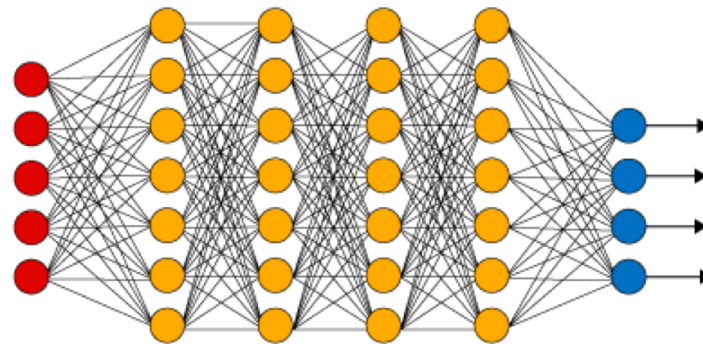
Neural Network for Symbolic Regression:

(1) **Neural Network is the function**



Neural Network for Symbolic Regression: (1) Neural Network is the function

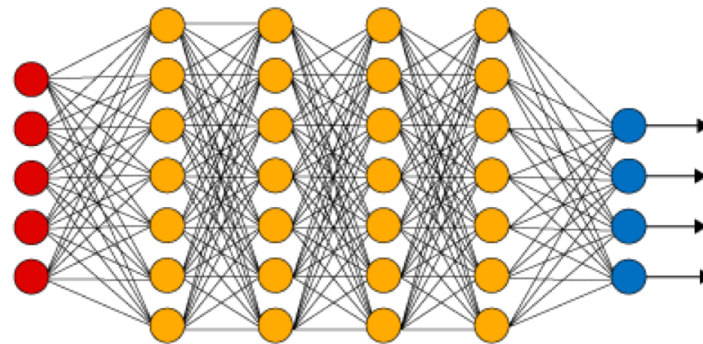
A neural network is an acyclic computation graph.



Idea: Let's use a neural network that "contains" the mystery function's computation graph.
We train the neural network to get the coefficients in the mystery function.

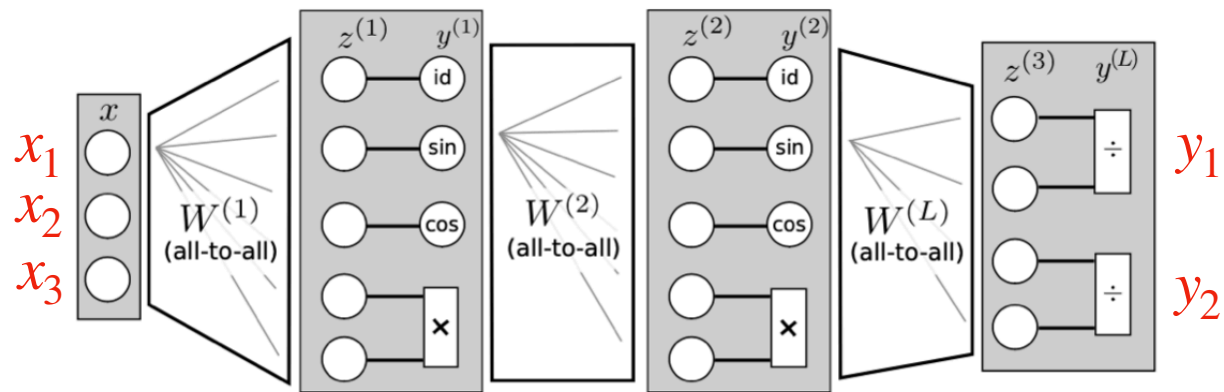
Neural Network for Symbolic Regression: (1) Neural Network is the function

A neural network is an acyclic computation graph.

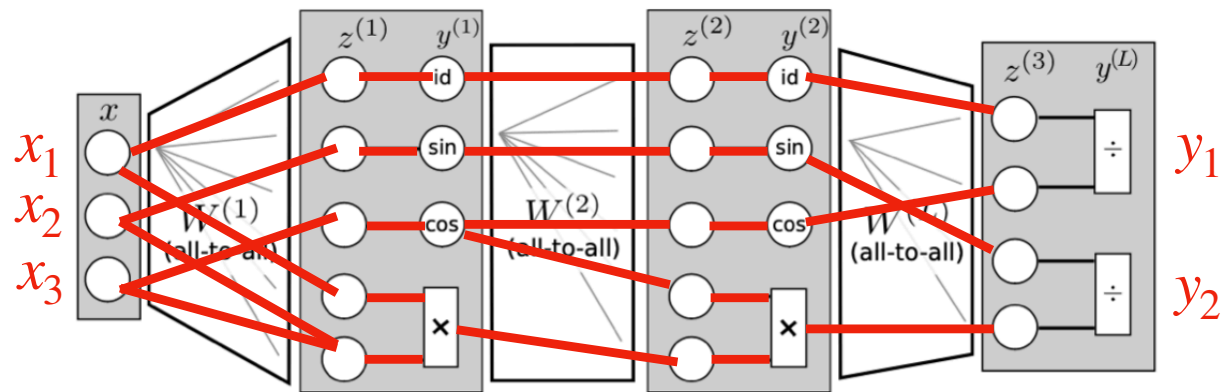


Idea: Let's use a neural network that “contains” the mystery function's computation graph.
We train the neural network to get the coefficients in the mystery function.

Neural Network for Symbolic Regression: (1) Neural Network is the function



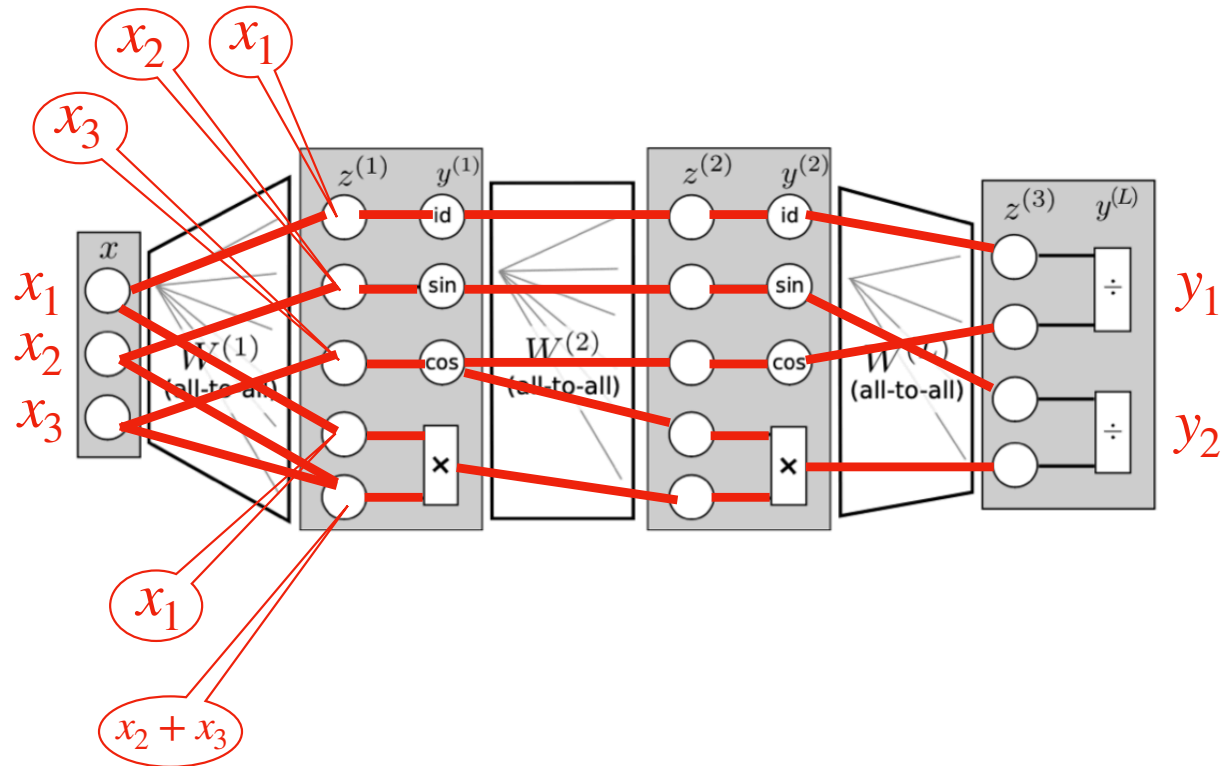
Neural Network for Symbolic Regression: (1) Neural Network is the function



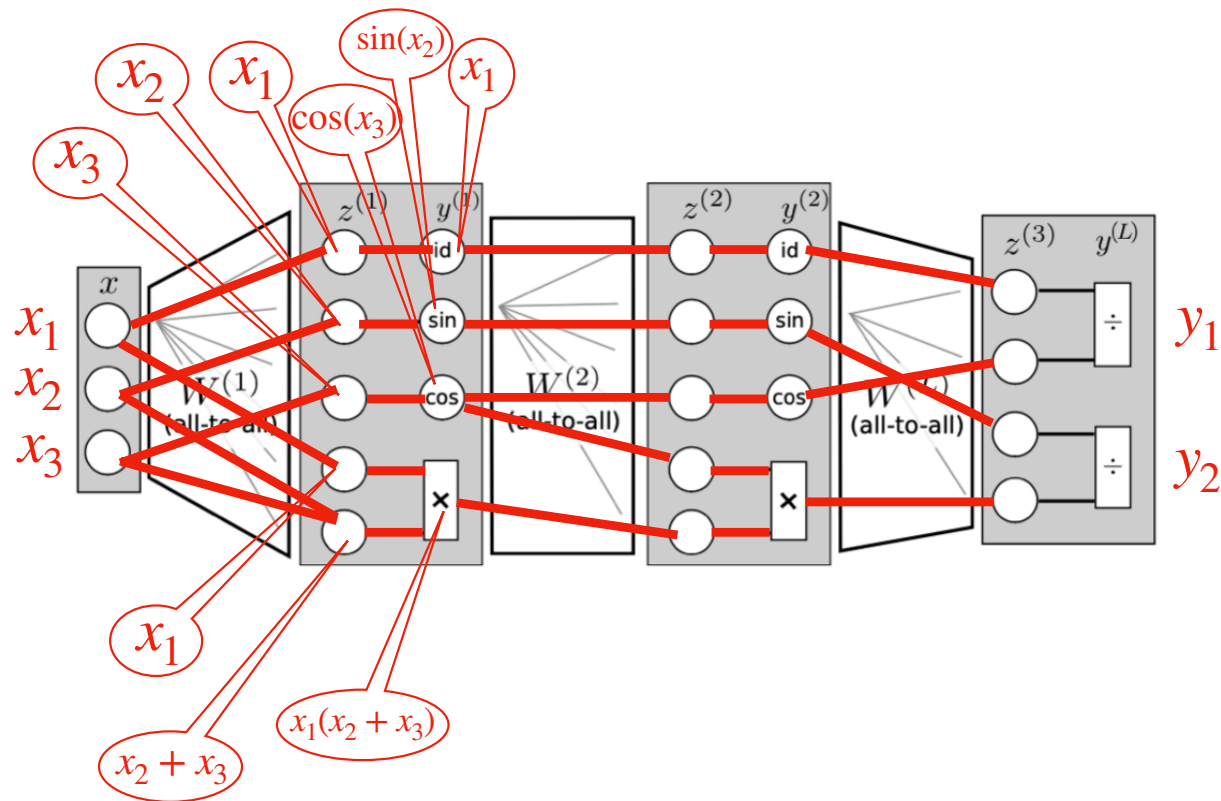
Train the above neural network.

For simplicity, assume after training, red edges have weight 1, other edges have weight 0.

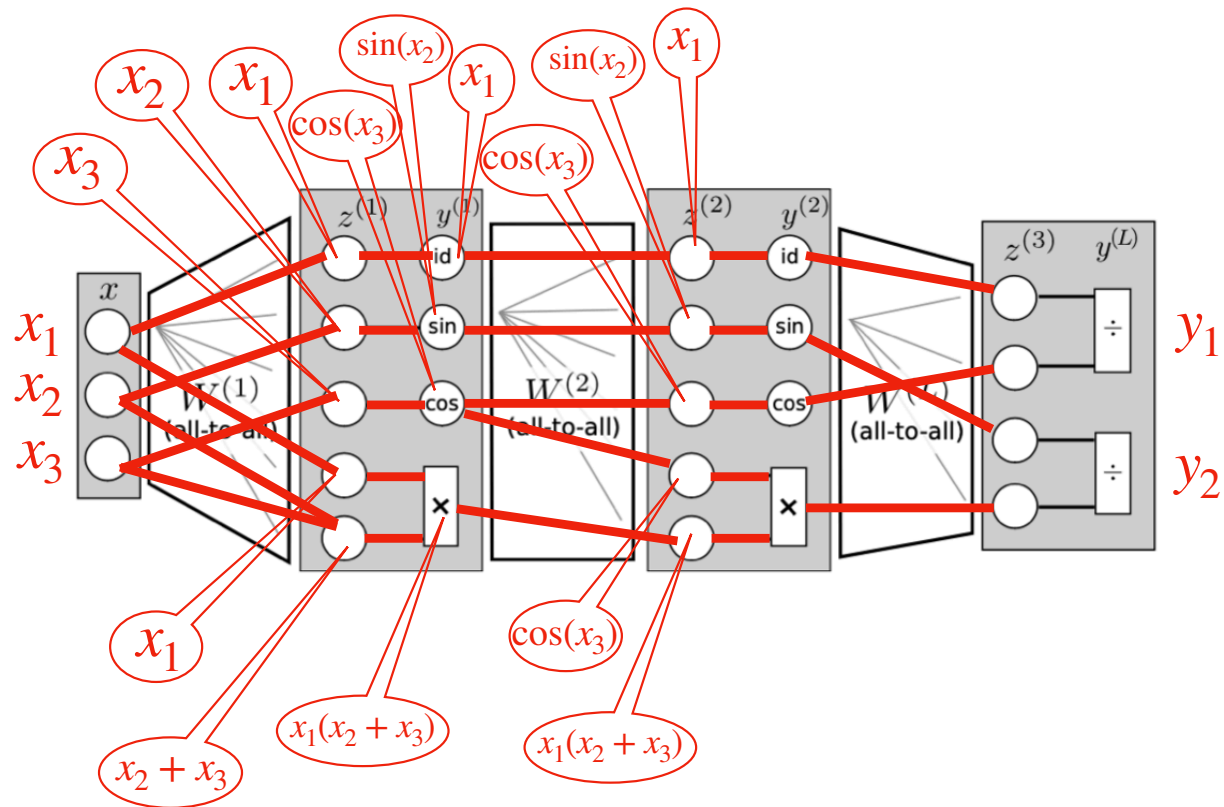
Neural Network for Symbolic Regression: (1) Neural Network is the function



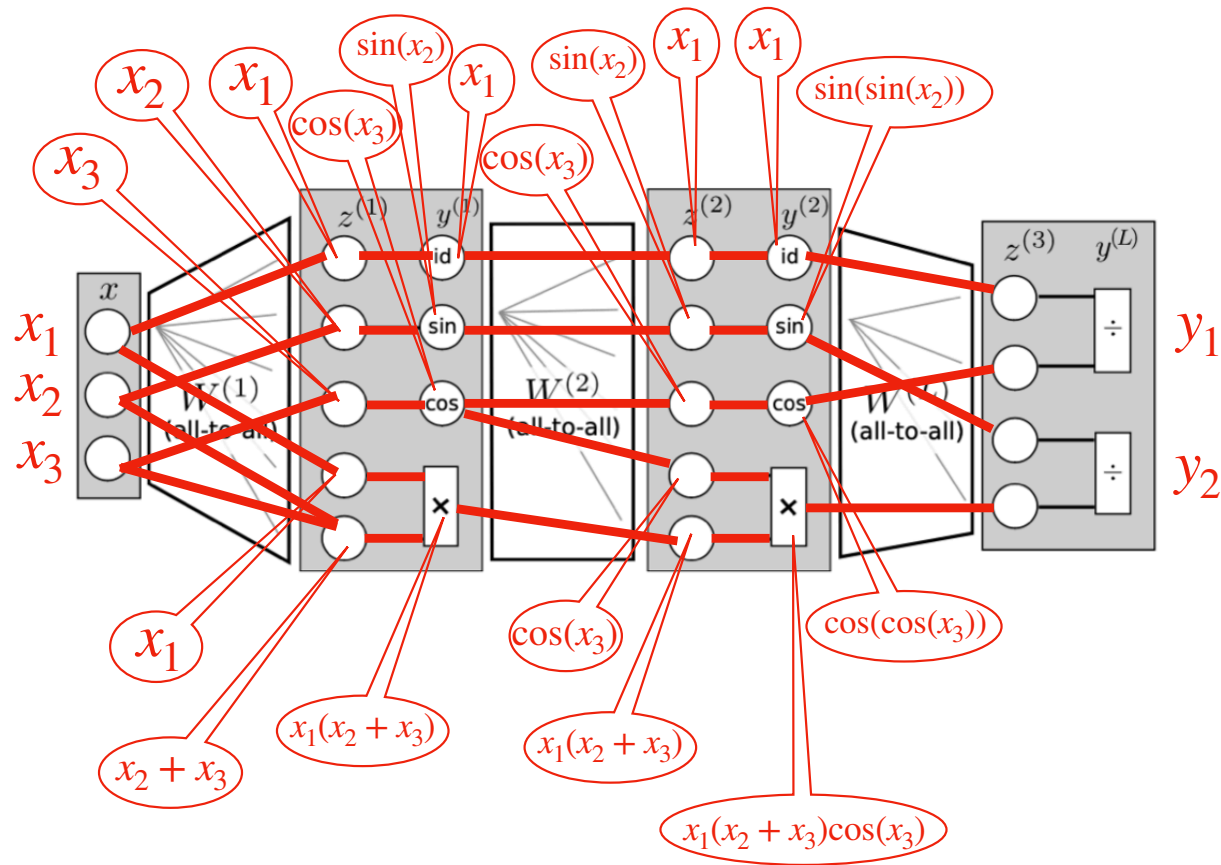
Neural Network for Symbolic Regression: (1) Neural Network is the function



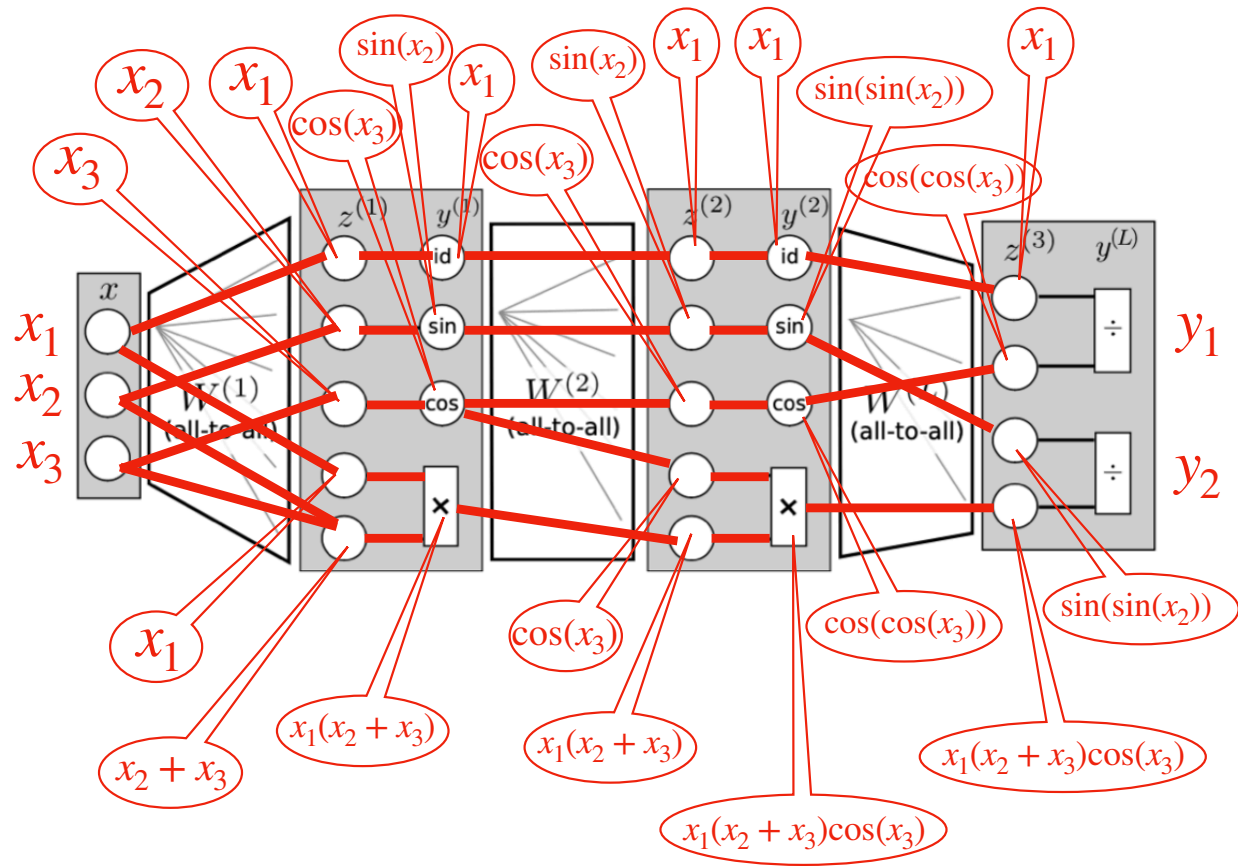
Neural Network for Symbolic Regression: (1) Neural Network is the function



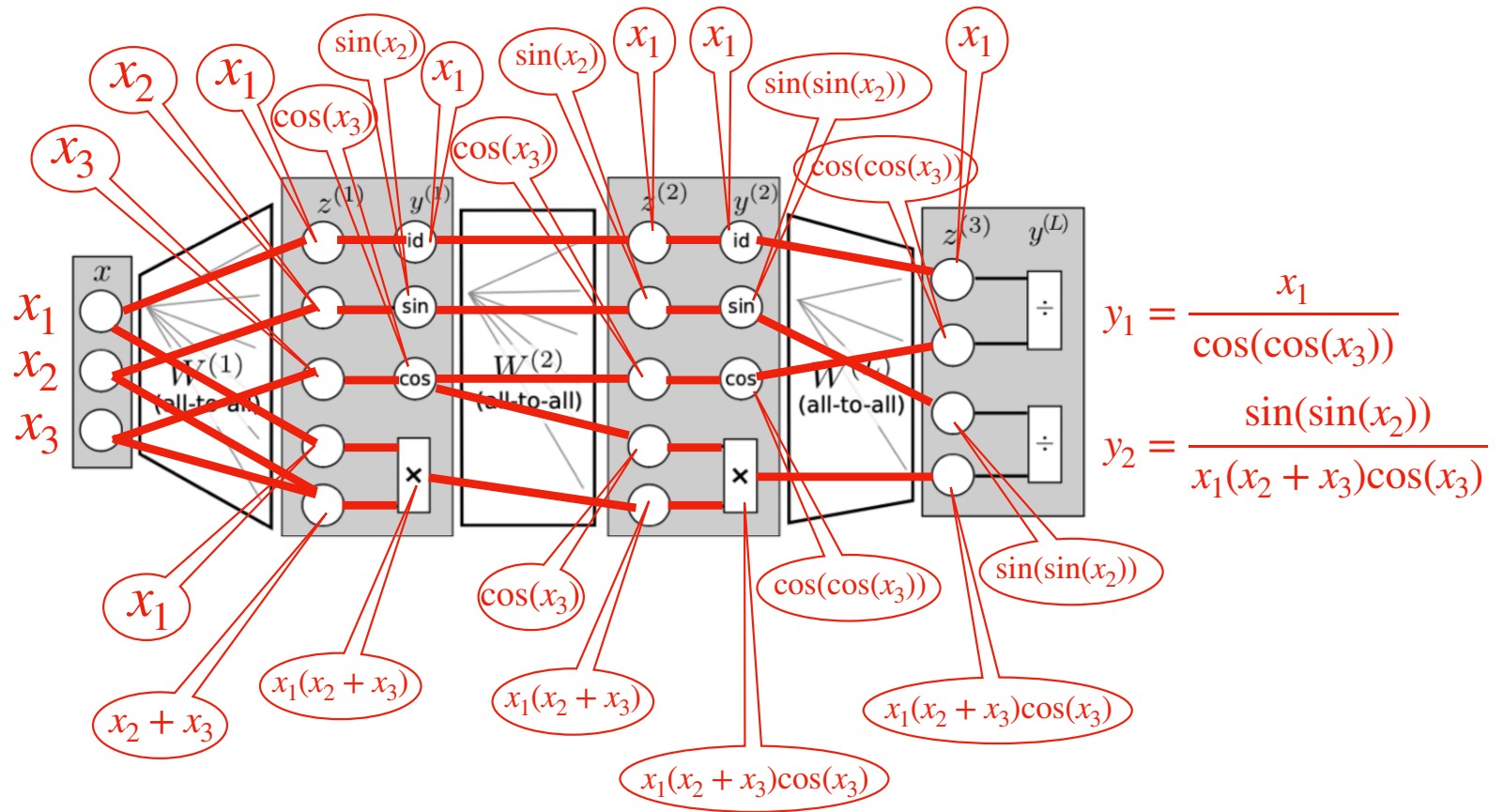
Neural Network for Symbolic Regression: (1) Neural Network is the function



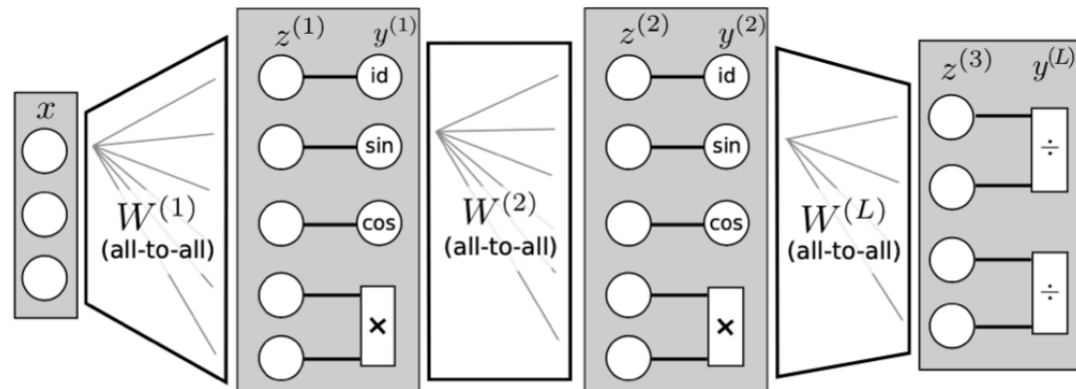
Neural Network for Symbolic Regression: (1) Neural Network is the function



Neural Network for Symbolic Regression: (1) Neural Network is the function



Neural Network for Symbolic Regression: (1) Neural Network is the function



EQL \div

A network for equation learning that can handle divisions as well as techniques to keep training stable.

Neural Network for Symbolic Regression: (1) Neural Network is the function

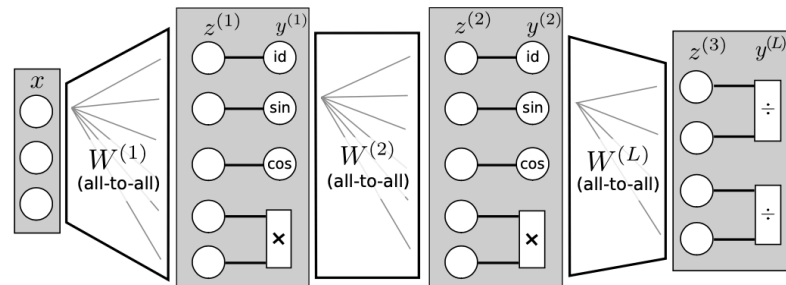


Figure 1. Network architecture of the proposed improved Equation Learner EQL⁺ for 3 layers ($L = 3$) and one neuron per type ($u = 3, v = 1$). The new division operations are places in the final layer, see [Martius & Lampert \(2016\)](#) for the original model.

Challenge posed by the division activation function:

Any division a/b creates a pole at $b \rightarrow 0$ with an abrupt change in the convexity and diverging function value and its derivative.

Such a divergence is a serious problem for gradient based optimization methods.

A few simplification made in the paper:

1. Assume that $b > 0$ in practical systems.
2. Use division only in the output layer.

Paper. [Learning Equations for Extrapolation and Control](#), by Subham S. Sahoo, Christoph H. Lampert, and Georg Martius, 35th ICML, Sweden, PMLR 80, 2018.

Neural Network for Symbolic Regression: (1) Neural Network is the function

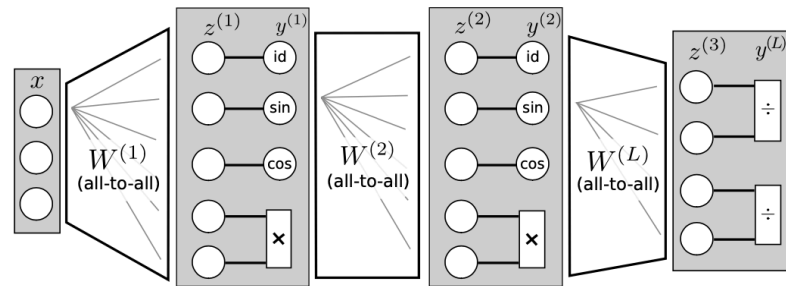


Figure 1. Network architecture of the proposed improved Equation Learner EQL⁺ for 3 layers ($L = 3$) and one neuron per type ($u = 3, v = 1$). The new division operations are places in the final layer, see [Martius & Lampert \(2016\)](#) for the original model.

Challenge posed by the division activation function:

Any division a/b creates a pole at $b \rightarrow 0$ with an abrupt change in the convexity and diverging function value and its derivative.

Such a divergence is a serious problem for gradient based optimization methods.

A few simplification made in the paper:

1. Assume that $b > 0$ in practical systems.
2. Use division only in the output layer.

Neural Network for Symbolic Regression: (1) Neural Network is the function

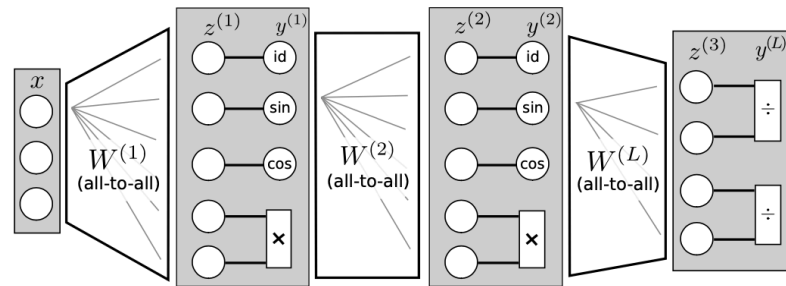


Figure 1. Network architecture of the proposed improved Equation Learner EQL⁺ for 3 layers ($L = 3$) and one neuron per type ($u = 3, v = 1$). The new division operations are places in the final layer, see [Martius & Lampert \(2016\)](#) for the original model.

Challenge posed by the division activation function:

Any division a/b creates a pole at $b \rightarrow 0$ with an abrupt change in the convexity and diverging function value and its derivative.

Such a divergence is a serious problem for gradient based optimization methods.

A few simplification made in the paper:

1. Assume that $b > 0$ in practical systems.
2. Use division only in the output layer.

Neural Network for Symbolic Regression: (1) Neural Network is the function

The division-activation function is given by $h^\theta(a, b) = \begin{cases} \frac{a}{b} & \text{if } b > \theta \\ 0 & \text{otherwise} \end{cases}$, where $\theta \geq 0$ is a threshold.

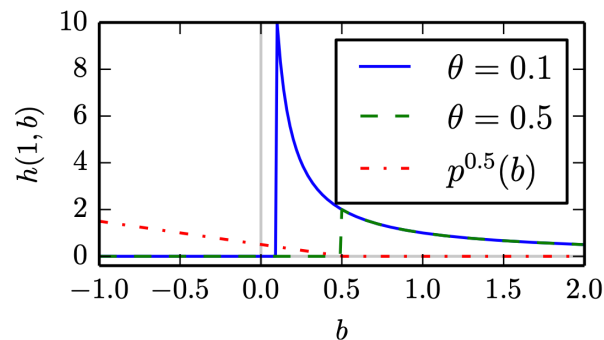


Figure 2. Regularized division function $h^\theta(a, b)$ and the associated penalty term $p^\theta(b)$. The penalty is linearly increasing for function values $b < \theta$ outside the desired input values.

Using $h^\theta = 0$ as the value when the denominator is below θ (forbidden values of b) sets the gradient to zero, avoiding misleading parameter updates.

Paper. Learning Equations for Extrapolation and Control, by Subham S. Sahoo, Christoph H. Lampert, and Georg Martius, 35th ICML, Sweden, PMLR 80, 2018.

Neural Network for Symbolic Regression: (1) Neural Network is the function

The division-activation function is given by
$$h^\theta(a, b) = \begin{cases} \frac{a}{b} & \text{if } b > \theta \\ 0 & \text{otherwise} \end{cases}, \text{ where } \theta \geq 0 \text{ is a threshold.}$$

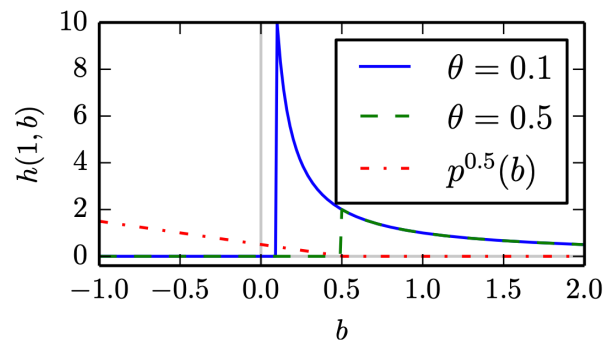


Figure 2. Regularized division function $h^\theta(a, b)$ and the associated penalty term $p^\theta(b)$. The penalty is linearly increasing for function values $b < \theta$ outside the desired input values.

Using $h^\theta = 0$ as the value when the denominator is below θ (forbidden values of b) sets the gradient to zero, avoiding misleading parameter updates.

Penalty term

To steer the network away from negative values of the denominator, we add a cost term to the training objective that penalizes “forbidden” inputs to each division unit:

$$p^\theta(b) = \max(\theta - b, 0)$$

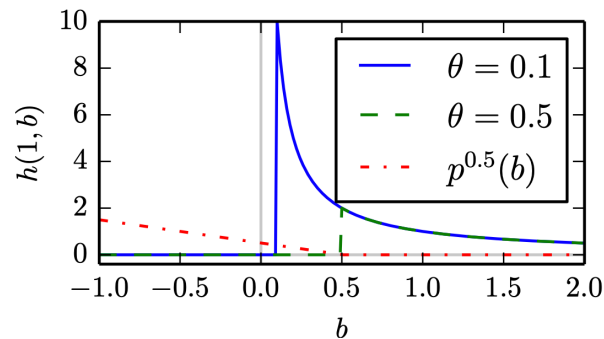


Figure 2. Regularized division function $h^\theta(a, b)$ and the associated penalty term $p^\theta(b)$. The penalty is linearly increasing for function values $b < \theta$ outside the desired input values.

Penalty Epochs

While the above approach prevents negative values in the denominator at training time (namely, for training samples), the right equation should not have negative denominators even for potential extrapolation data (namely, data outside of the region of training data).

Similarly, we would like to prevent that output values on future data having a very different magnitude than the observed outputs, as this could be a sign of overfitting (e.g., learning a polynomial of too high-degree).

To enforce this we introduce particular “penalty epochs”, which are injected at regular intervals (every 50 epochs) into training process.

Penalty Epochs

While the above approach prevents negative values in the denominator at training time (namely, for training samples), the right equation should not have negative denominators even for potential extrapolation data (namely, data outside of the region of training data).

Similarly, we would like to prevent that output values on future data having a very different magnitude than the observed outputs, as this could be a sign of overfitting (e.g., learning a polynomial of too high-degree).

To enforce this we introduce particular “penalty epochs”, which are injected at regular intervals (every 50 epochs) into training process.

Penalty Epochs

While the above approach prevents negative values in the denominator at training time (namely, for training samples), the right equation should not have negative denominators even for potential extrapolation data (namely, data outside of the region of training data).

Similarly, we would like to prevent that output values on future data having a very different magnitude than the observed outputs, as this could be a sign of overfitting (e.g., learning a polynomial of too high-degree).

To enforce this we introduce particular “penalty epochs”, which are injected at regular intervals (every 50 epochs) into training process.

Penalty Epochs

During a penalty epoch, we randomly sample N input data points in the expected “test range” (including extrapolation region) without labels and the network is trained using the cost

$$L^{\text{Penalty}} = P^{\theta} + P^{\text{bound}}$$

where the latter is given by

$$P^{\text{bound}} = \sum_{i=1}^N \sum_{j=1}^n \max(y_j^L(x_i) - B, 0) + \max(-y_j^L(x_i) - B, 0)$$

Basically, the penalty P^{bound} is > 0 if any output by the network is outside the range $[-B, B]$. The value B reflects the maximal desired output value.

Curriculum Training (for division activation function)

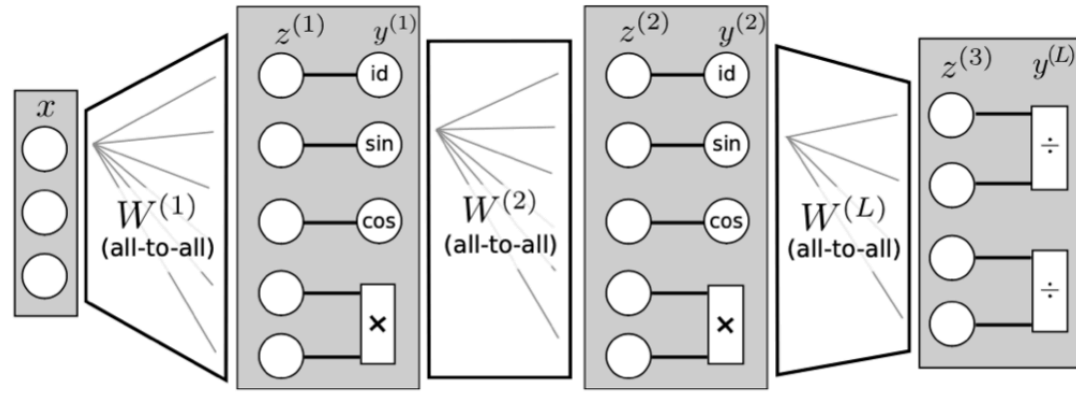
The division-activation function is given by
$$h^\theta(a, b) = \begin{cases} \frac{a}{b} & \text{if } b > \theta \\ 0 & \text{otherwise} \end{cases}$$

We let θ decrease with epoch t as
$$\theta(t) = \frac{1}{\sqrt{t+1}}$$

It helps the network learn the equation more accurately in the end. For validation and testing, we use

$$\theta = 10^{-4}$$

Network training



The network is fully differentiable, which allows us to train it in an end-to-end fashion using back propagation. The objective is Lasso-like:

$$L = \frac{1}{N} \sum_{i=1}^N (f(x_i) - y_i)^2 + \lambda \sum_{l=1}^L |w^{(l)}| + P^\theta$$

MSE loss

L1 regularization
to get a
simple function

Cost for small
and negative
denominators in
division

Network training

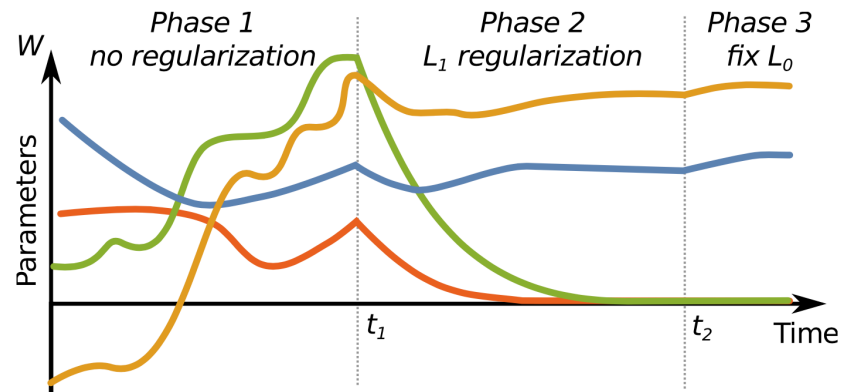
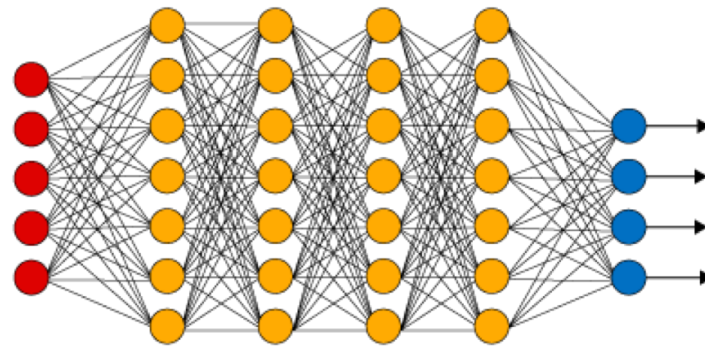
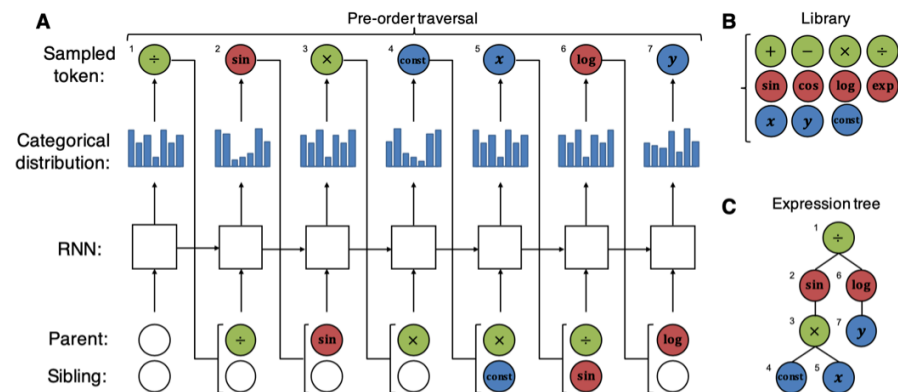


Figure 3. Regularization phases: there is no regularization in the first phase ($t < t_1$) where the weights can move freely, followed by a normal L_1 phase ($t_1 \leq t < t_2$) where many weights go to zero, followed by a phase ($t_2 \leq t$) that fixes the L_0 norm by keeping small weights at zero and allowing all other weights to go to their correct value. Figure adapted from [Martius & Lampert \(2016\)](#).

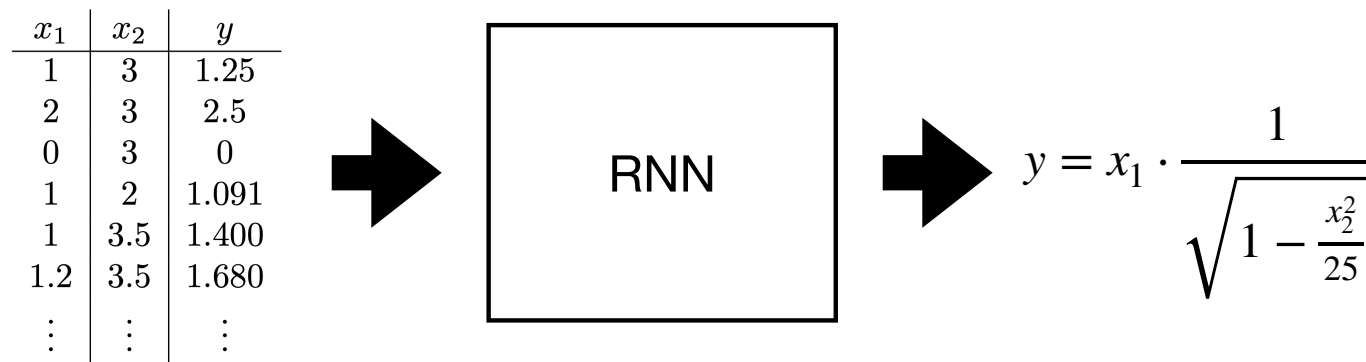
Neural Network for Symbolic Regression



Neural Network for Symbolic Regression: (2) Neural Network outputs the function



Neural Network for Symbolic Regression: (2) Neural Network outputs the function



Neural Network for Symbolic Regression: (2) Neural Network outputs the function

We present “Deep Symbolic Regression” (DSR), a gradient-based approach for symbolic regression based on reinforcement learning.

In DSR, a recurrent network (RNN) emits a distribution over mathematical expressions. Expressions are sampled from the distribution, instantiated, and evaluated based on their fitness to the dataset.

This fitness is used as the reward signal to train the RNN using a risk-seeking policy gradient algorithm.

As training proceeds, the RNN adjusts the likelihood of an expression relative to its reward, assigning higher probabilities to better expressions.

DSR is autoregressive, meaning each token is conditioned on the previously sampled token.

DSR exploits the hierarchical nature of trees by providing the parent and sibling as inputs to the RNN.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function

We present “Deep Symbolic Regression” (DSR), a gradient-based approach for symbolic regression based on reinforcement learning.

In DSR, a recurrent network (RNN) emits a distribution over mathematical expressions. Expressions are sampled from the distribution, instantiated, and evaluated based on their fitness to the dataset.

This fitness is used as the reward signal to train the RNN using a risk-seeking policy gradient algorithm.

As training proceeds, the RNN adjusts the likelihood of an expression relative to its reward, assigning higher probabilities to better expressions.

DSR is autoregressive, meaning each token is conditioned on the previously sampled token.

DSR exploits the hierarchical nature of trees by providing the parent and sibling as inputs to the RNN.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function

We present “Deep Symbolic Regression” (DSR), a gradient-based approach for symbolic regression based on reinforcement learning.

In DSR, a recurrent network (RNN) emits a distribution over mathematical expressions. Expressions are sampled from the distribution, instantiated, and evaluated based on their fitness to the dataset.

This fitness is used as the reward signal to train the RNN using a risk-seeking policy gradient algorithm.

As training proceeds, the RNN adjusts the likelihood of an expression relative to its reward, assigning higher probabilities to better expressions.

DSR is autoregressive, meaning each token is conditioned on the previously sampled token.

DSR exploits the hierarchical nature of trees by providing the parent and sibling as inputs to the RNN.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function

We present “Deep Symbolic Regression” (DSR), a gradient-based approach for symbolic regression based on reinforcement learning.

In DSR, a recurrent network (RNN) emits a distribution over mathematical expressions. Expressions are sampled from the distribution, instantiated, and evaluated based on their fitness to the dataset.

This fitness is used as the reward signal to train the RNN using a risk-seeking policy gradient algorithm.

As training proceeds, the RNN adjusts the likelihood of an expression relative to its reward, assigning higher probabilities to better expressions.

DSR is autoregressive, meaning each token is conditioned on the previously sampled token.

DSR exploits the hierarchical nature of trees by providing the parent and sibling as inputs to the RNN.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function

We present “Deep Symbolic Regression” (DSR), a gradient-based approach for symbolic regression based on reinforcement learning.

In DSR, a recurrent network (RNN) emits a distribution over mathematical expressions. Expressions are sampled from the distribution, instantiated, and evaluated based on their fitness to the dataset.

This fitness is used as the reward signal to train the RNN using a risk-seeking policy gradient algorithm.

As training proceeds, the RNN adjusts the likelihood of an expression relative to its reward, assigning higher probabilities to better expressions.

DSR is autoregressive, meaning each token is conditioned on the previously sampled token.

DSR exploits the hierarchical nature of trees by providing the parent and sibling as inputs to the RNN.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function

We present “Deep Symbolic Regression” (DSR), a gradient-based approach for symbolic regression based on reinforcement learning.

In DSR, a recurrent network (RNN) emits a distribution over mathematical expressions. Expressions are sampled from the distribution, instantiated, and evaluated based on their fitness to the dataset.

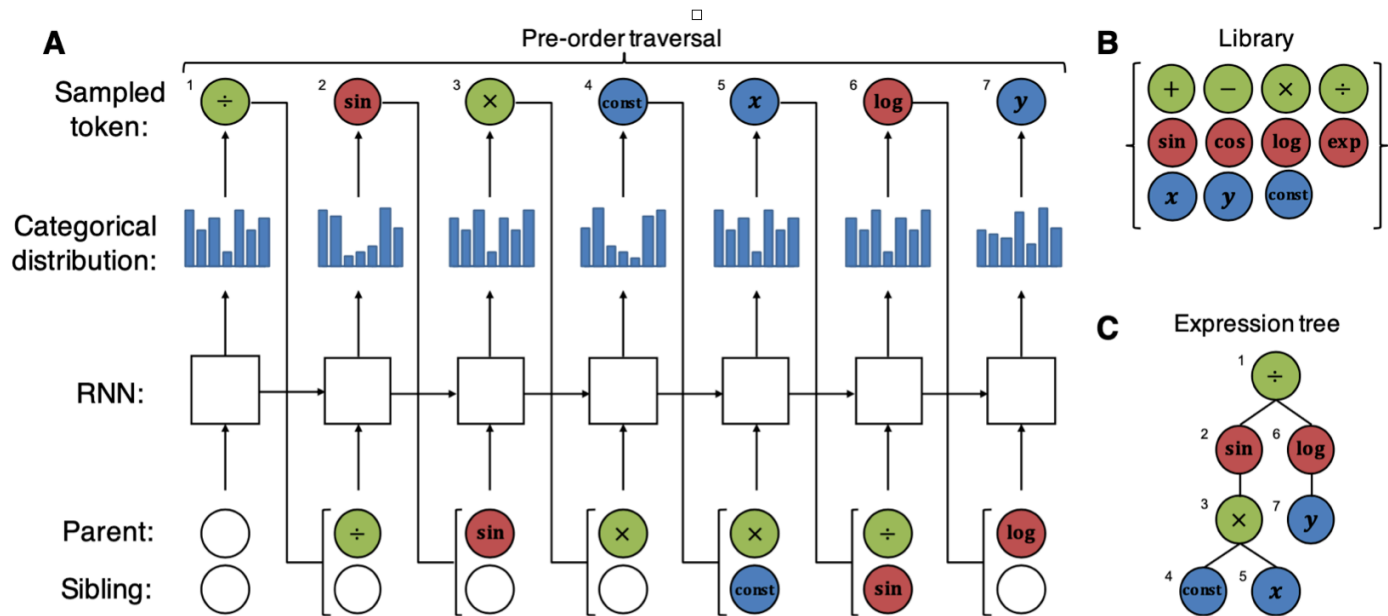
This fitness is used as the reward signal to train the RNN using a risk-seeking policy gradient algorithm.

As training proceeds, the RNN adjusts the likelihood of an expression relative to its reward, assigning higher probabilities to better expressions.

DSR is autoregressive, meaning each token is conditioned on the previously sampled token.

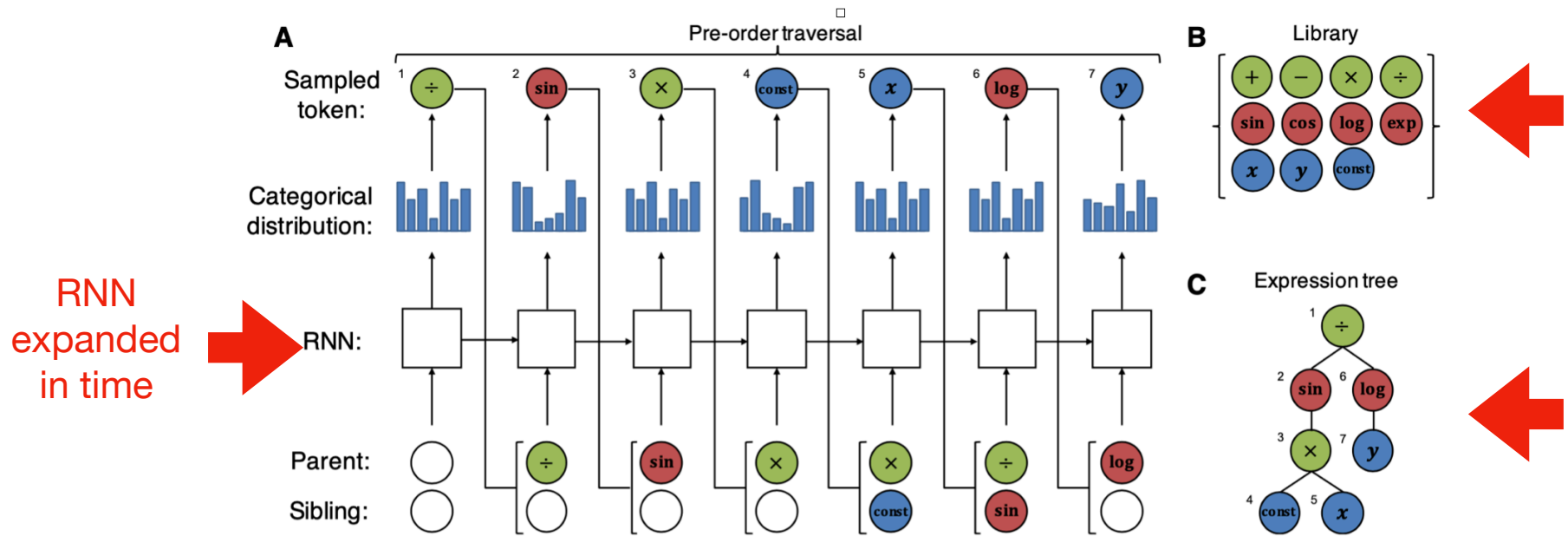
DSR exploits the hierarchical nature of trees by providing the parent and sibling as inputs to the RNN.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



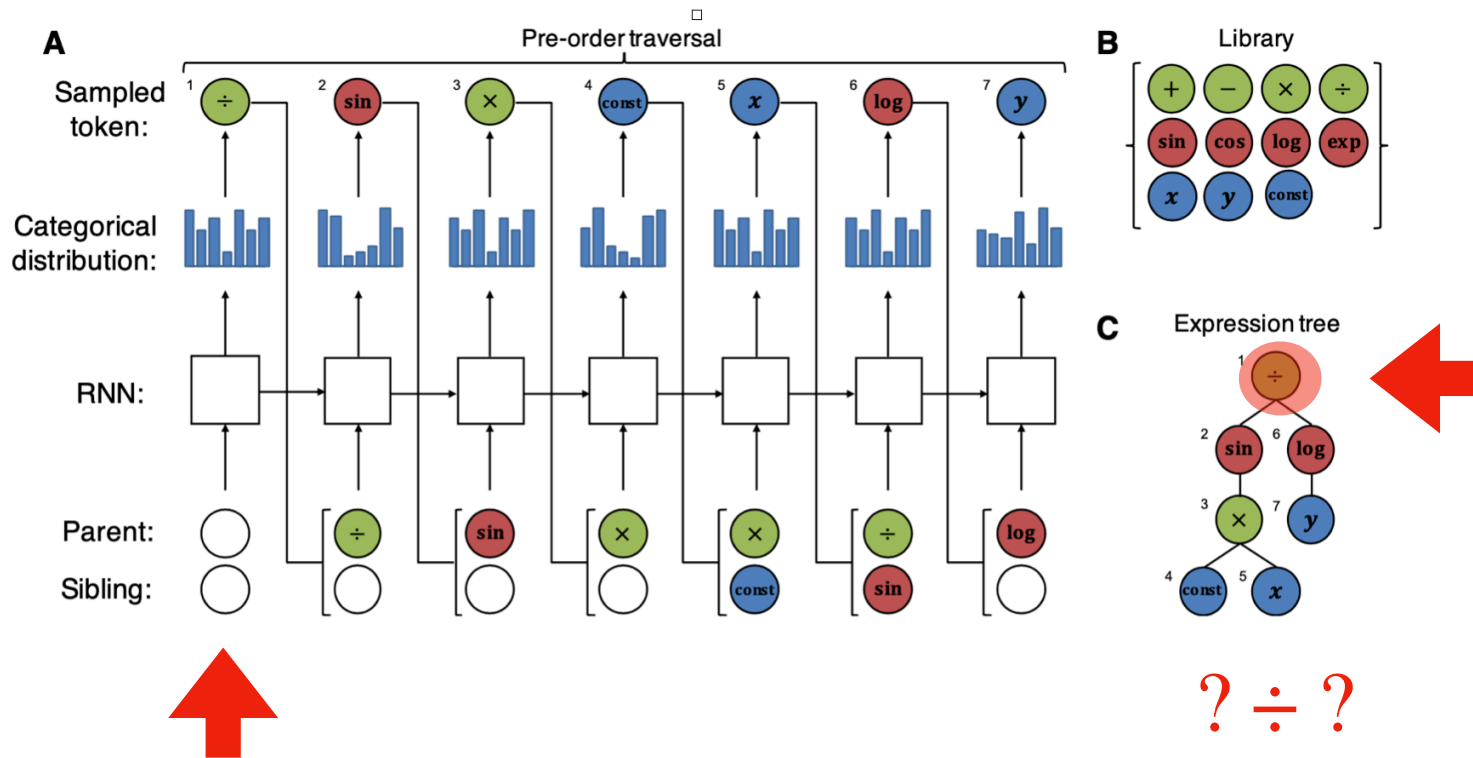
Paper. [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



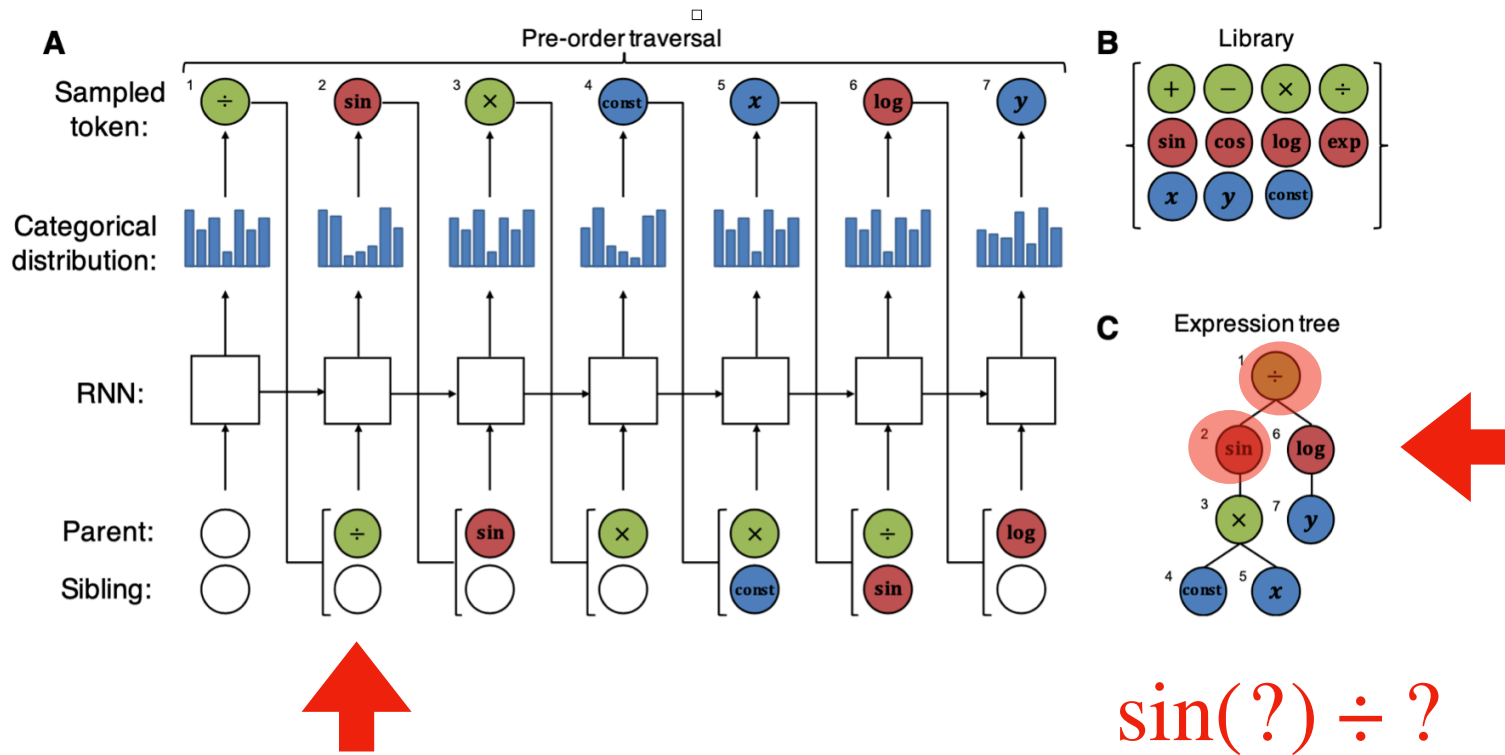
Paper. [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



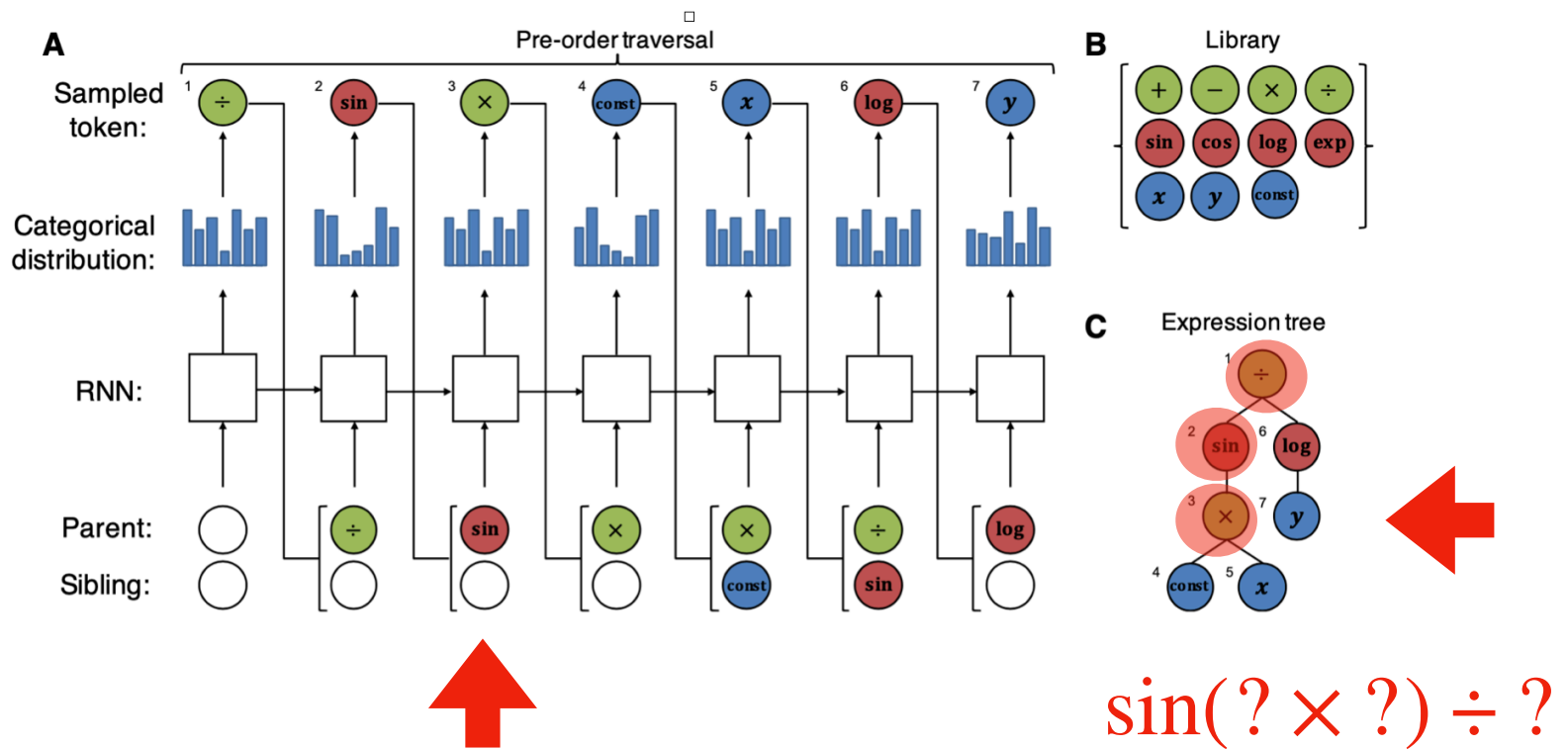
Paper: [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



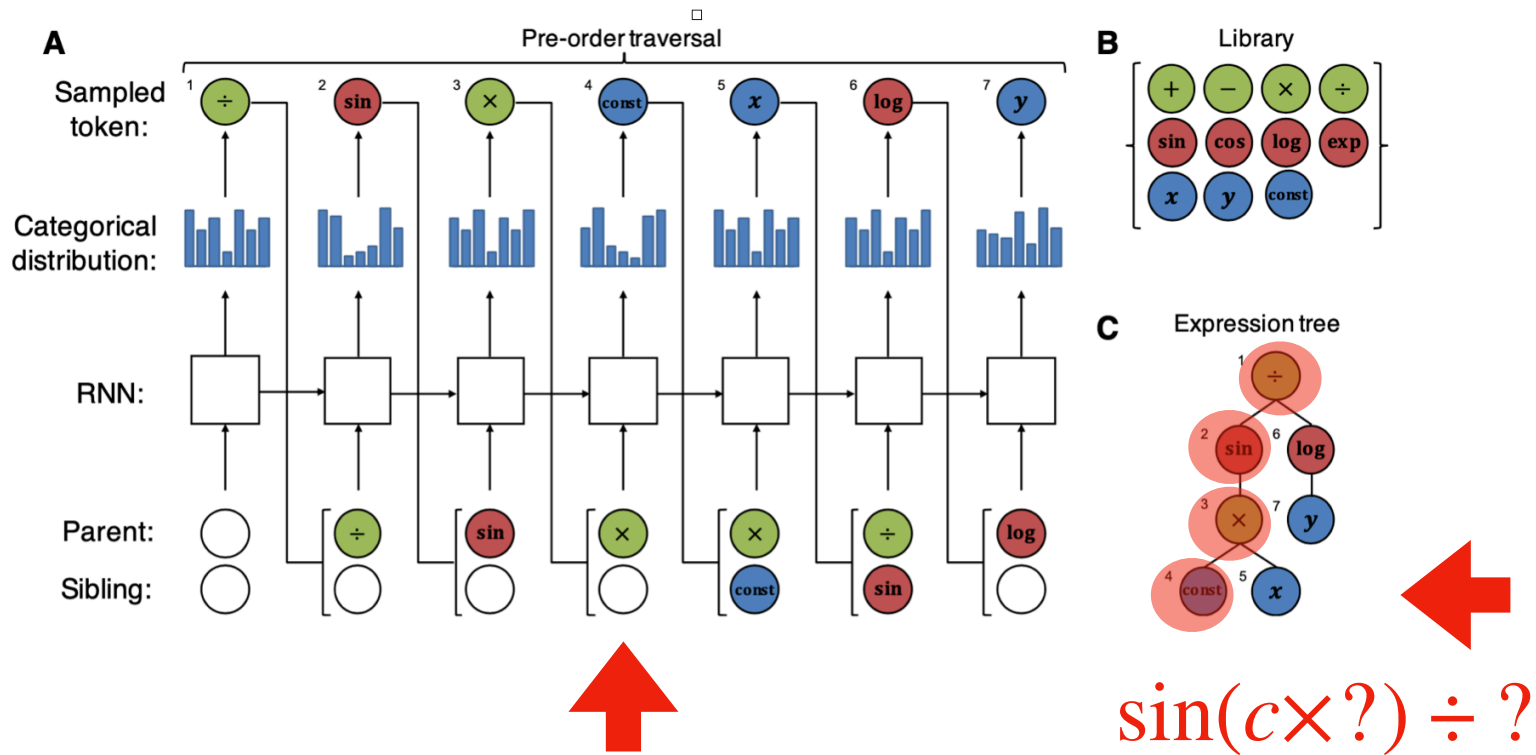
Paper: [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



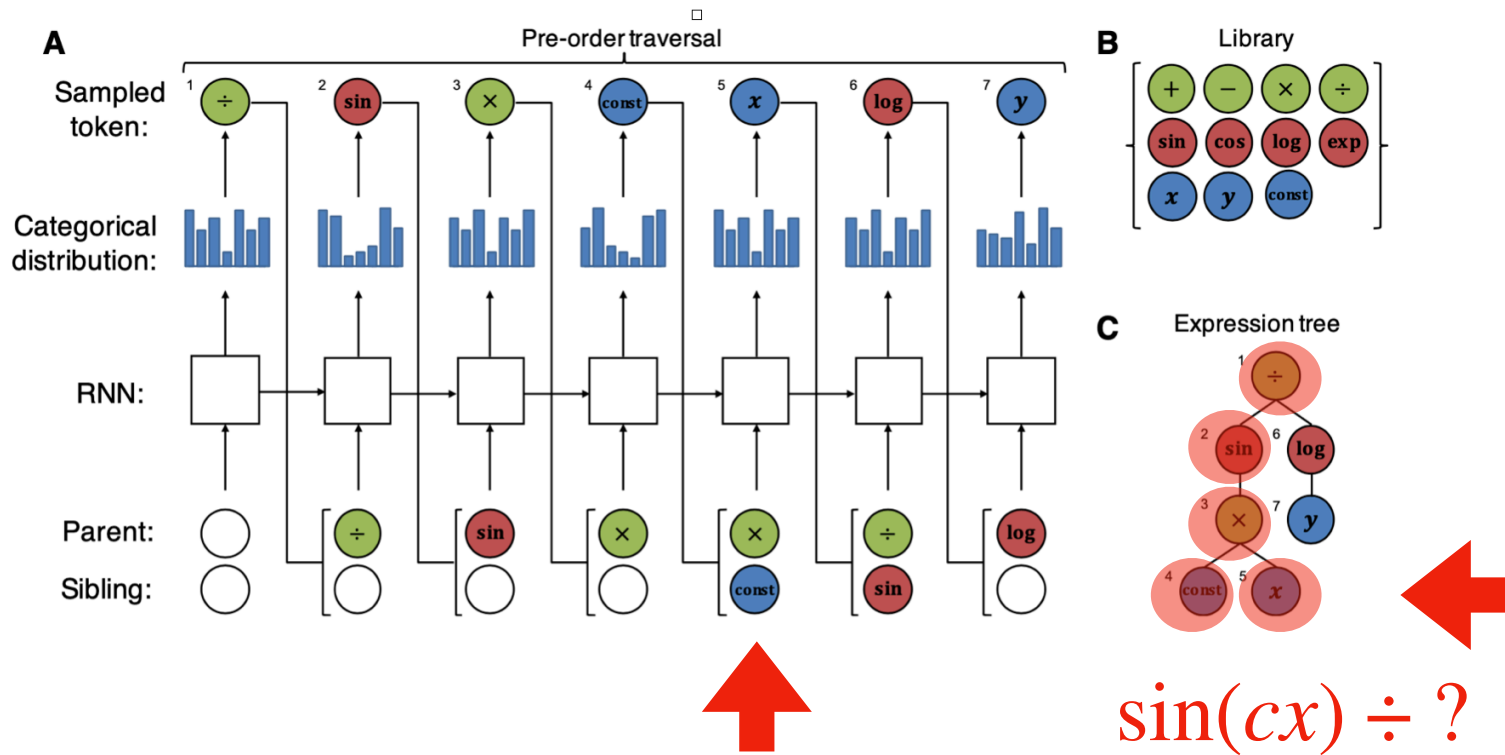
Paper: [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



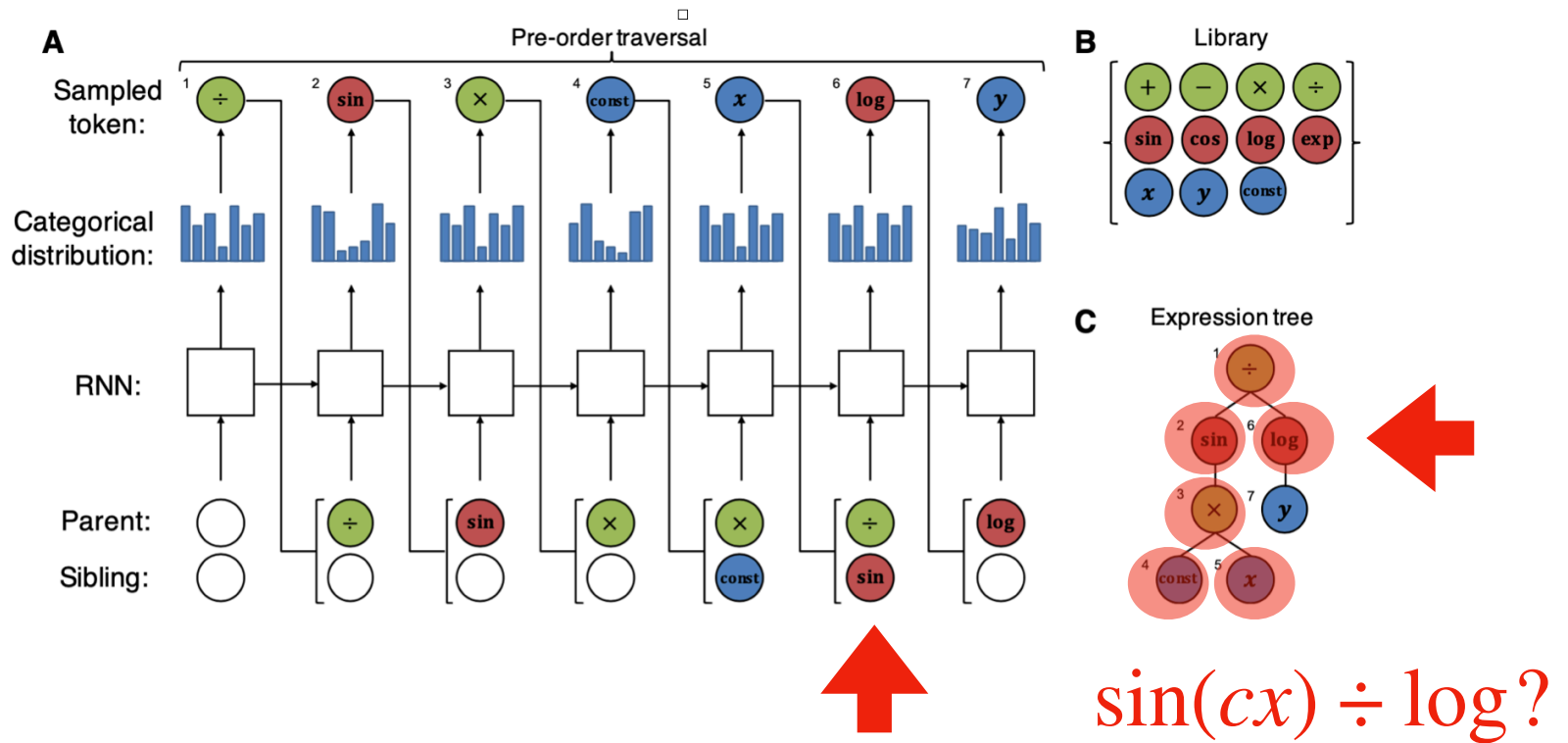
Paper: [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



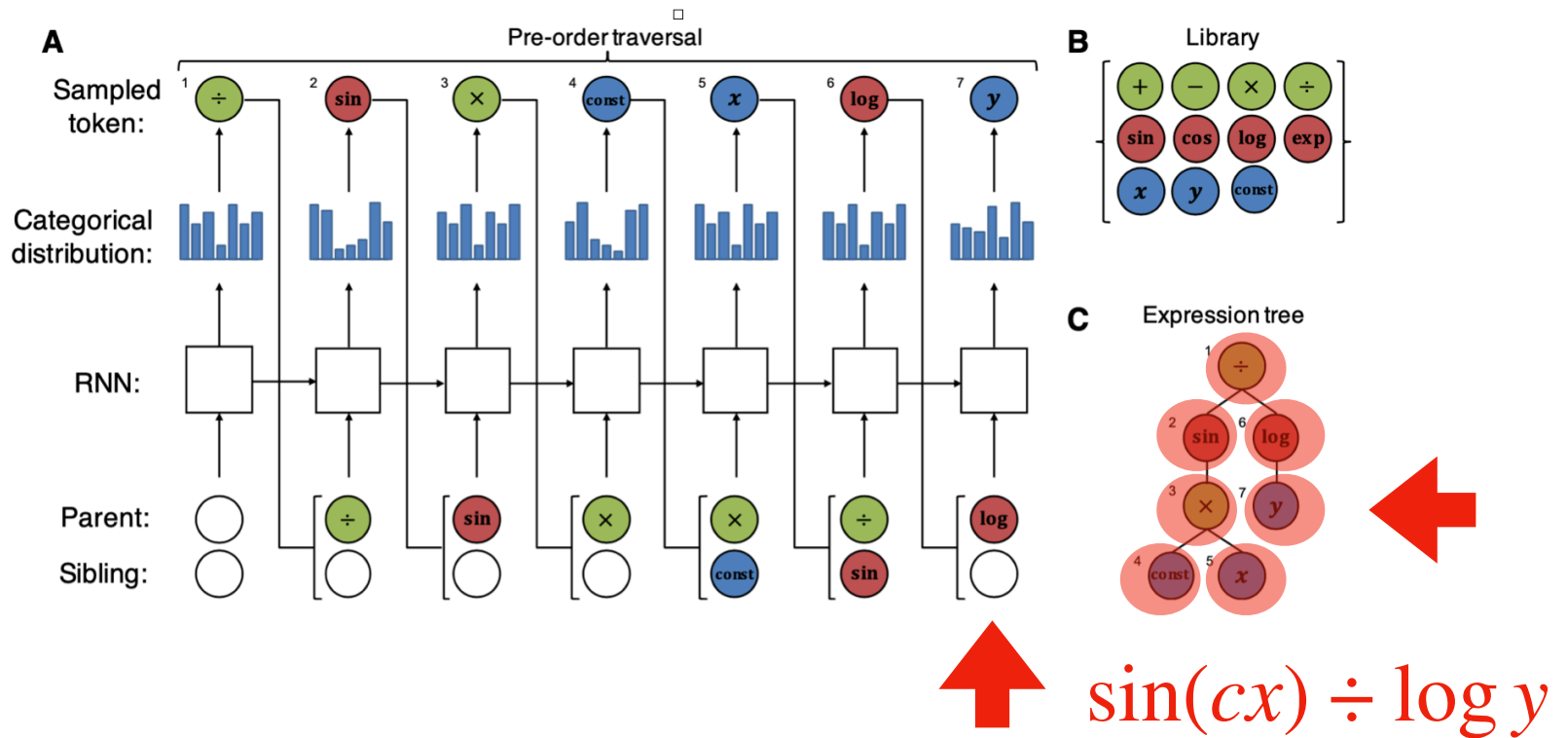
Paper: [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



Paper: [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Neural Network for Symbolic Regression: (2) Neural Network outputs the function



Paper: [Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients](#),
 By Brenden K. Petersen, Mikel Landajuela Larma, T. Nathan Mundhenk, Claudio Santiago, Sookyung Kim and Joanne T. Kim, at ICLR 2021.

Constraining the search space

It is straightforward to apply a priori constraints to reduce the search space. Here are some plausible constraints:

- 1) the children of an operator should not all be constants.
- 2) The child of a unary operator should not be the inverse of that operator, e.g., $\log(\exp(x))$ is not allowed.

We apply these constraints concurrently with autoregressive sampling by zeroing out the probabilities of selecting tokens that violate a constraint.

Reward function for reinforcement learning

A standard fitness measure in symbolic regression is normalized root-mean-square error (NRMSE), the root-mean-square error normalized by the standard deviation of the target values, σ_y .

That is, given a dataset (X,y) of size n and candidate expression f ,

$$NRMSE = \frac{1}{\sigma_y} \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - f(X_i))^2}$$

Reward function:

$$R(\tau) = \frac{1}{1 + NRMSE}$$

Constant optimization

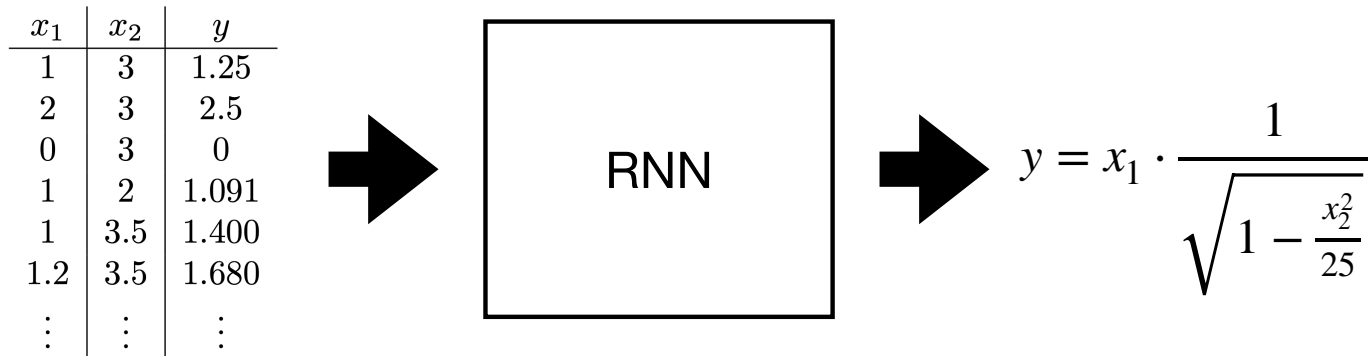
If the Library L includes constant token, sampled expressions may include several constant placeholders. These can be viewed as parameters β of the symbolic expression, which we optimize by maximizing the reward function

$$\beta^* = \arg \max_{\beta} R(\tau; \beta)$$

using a nonlinear optimization (nonlinear regression) algorithm, e.g., BFGS.

We perform this inner optimization loop for each sampled expression (namely, after a whole symbolic function is generated by the RNN) as part of the reward computation before performing each training step.

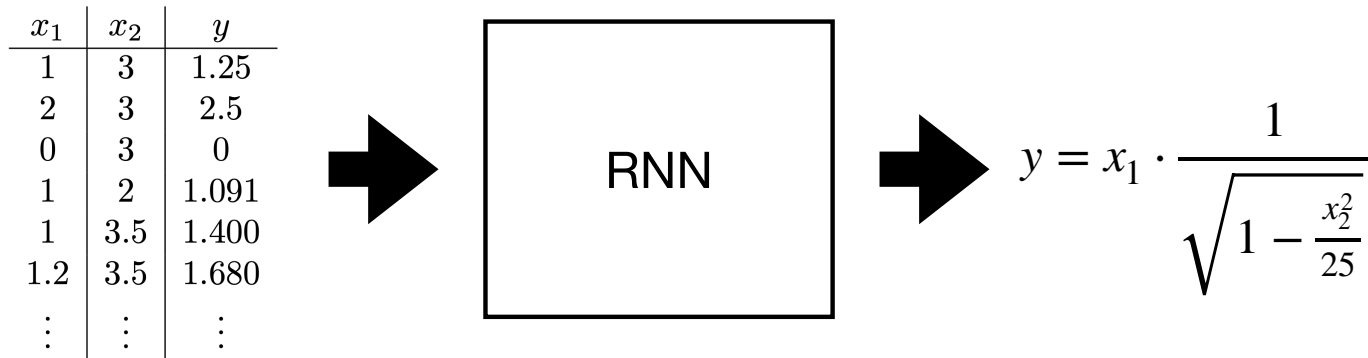
Training the RNN using Policy Gradients



Now that we have a distribution over mathematical expressions $p(\tau|\theta)$, where τ is a function and θ is the RNN's weights, we first consider the standard policy gradient objective to maximize

$$J_{std}(\theta) = E_{\tau \sim (\tau|\theta)} R(\tau)$$

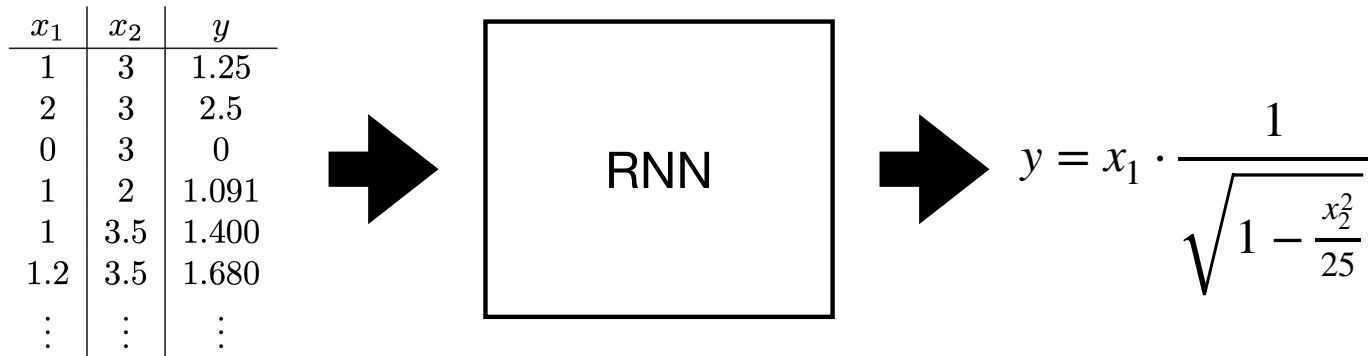
Training the RNN using Policy Gradients



The standard REINFORCE policy gradient can be used to maximize this expectation via gradient ascent:

$$\nabla_{\theta} J_{std}(\theta) = \nabla_{\theta} E_{\tau \sim (\tau|\theta)} R(\tau) = E_{\tau \sim p(\tau|\theta)} R(\tau) \nabla_{\theta} \log p(\tau|\theta)$$

Training the RNN using Policy Gradients



This result allows one to estimate the expectation using samples from the distribution. Specifically, an unbiased estimate can be obtained by computing the sample mean over a batch of N sampled expressions

$$\nabla_{\theta} J_{std}(\theta) \approx \frac{1}{N} \sum_{i=1}^N R(\tau^i) \nabla_{\theta} \log p(\tau^i | \theta)$$

This is an unbiased gradient estimate, but in practice it has high variance. To reduce variance, it is common to subtract a baseline function b from the reward. As long as the baseline is not a function of the current batch of expressions, the gradient estimate is still unbiased. Common choices of baseline functions are a moving average of rewards or an estimate of the value function.

Risk-seeking policy gradient

Ideally, the RNN should always output the unique correct symbolic expression. But in practice, the RNN can output a set of candidate symbolic expressions.

We want to maximize the expected reward of the (one or a few) best expressions, not the average reward of all output expressions. (This is especially important when there are too many output expressions.)

Idea: instead of maximizing the expected reward of all output expressions (and hoping they will eventually all converge to the correct expressions), let's focus only on the best few output expressions and maximize their expected reward, and dismiss the other less-optimal expressions.

That is, we propose an alternative objective that focuses on learning only on “maximizing best-case performance”.

Risk-seeking policy gradient

Ideally, the RNN should always output the unique correct symbolic expression. But in practice, the RNN can output a set of candidate symbolic expressions.

We want to maximize the expected reward of the (one or a few) best expressions, not the average reward of all output expressions. (This is especially important when there are too many output expressions.)

Idea: instead of maximizing the expected reward of all output expressions (and hoping they will eventually all converge to the correct expressions), let's focus only on the best few output expressions and maximize their expected reward, and dismiss the other less-optimal expressions.

That is, we propose an alternative objective that focuses on learning only on “maximizing best-case performance”.

Risk-seeking policy gradient

Ideally, the RNN should always output the unique correct symbolic expression. But in practice, the RNN can output a set of candidate symbolic expressions.

We want to maximize the expected reward of the (one or a few) best expressions, not the average reward of all output expressions. (This is especially important when there are too many output expressions.)

Idea: instead of maximizing the expected reward of all output expressions (and hoping they will eventually all converge to the correct expressions), let's focus only on the best few output expressions and maximize their expected reward, and dismiss the other less-optimal expressions.

That is, we propose an alternative objective that focuses on learning only on “maximizing best-case performance”.

Risk-seeking policy gradient

Ideally, the RNN should always output the unique correct symbolic expression. But in practice, the RNN can output a set of candidate symbolic expressions.

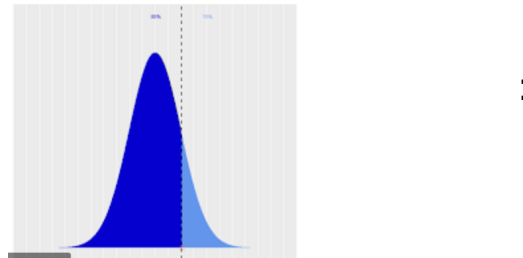
We want to maximize the expected reward of the (one or a few) best expressions, not the average reward of all output expressions. (This is especially important when there are too many output expressions.)

Idea: instead of maximizing the expected reward of all output expressions (and hoping they will eventually all converge to the correct expressions), let's focus only on the best few output expressions and maximize their expected reward, and dismiss the other less-optimal expressions.

That is, we propose an alternative objective that focuses on learning only on “maximizing best-case performance”.

Risk-seeking policy gradient

We first define $R_\epsilon(\theta)$ as the $(1 - \epsilon)$ -quantile of the distribution of the rewards under the current policy.



We then propose a new learning objective $J_{risk}(\theta; \epsilon)$ parameterized by ϵ :

$$J_{risk}(\theta; \epsilon) = E_{\tau \sim p(\tau|\theta)} [R(\tau) | R(\tau) \geq R_\epsilon(\theta)]$$

This objective aims to increase the reward of the top ϵ fraction of samples from the distribution, without regard for samples below that threshold. It aims to increase best-case performance at the expense of lower average-case performance.

Risk-seeking policy gradient

$$\nabla_{\theta} J_{risk}(\theta; \epsilon) = E_{\tau \sim p(\tau|\theta)} [(R(\tau) - R_{\epsilon}(\theta)) \nabla_{\theta} \log p(\tau|\theta) \mid R(\tau) \geq R_{\epsilon}(\theta)]$$

Monte Carlo estimate of the gradient from a batch of N samples:

$$\nabla_{\theta} J_{risk}(\theta; \epsilon) \approx \frac{1}{\epsilon N} \sum_{i=1}^N [R(\tau^{(i)}) - \tilde{R}_{\epsilon}(\theta)] \cdot 1_{R(\tau^{(i)}) \geq \tilde{R}_{\epsilon}(\theta)} \nabla_{\theta} \log p(\tau^{(i)}|\theta)$$

Evaluating DSR

Nguyen symbolic regression benchmark: a set of 12 commonly used benchmark expressions.

Benchmark	Expression	DSR
Nguyen-1	$x^3 + x^2 + x$	100%
Nguyen-2	$x^4 + x^3 + x^2 + x$	100%
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$	100%
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	100%
Nguyen-5	$\sin(x^2) \cos(x) - 1$	72%
Nguyen-6	$\sin(x) + \sin(x + x^2)$	100%
Nguyen-7	$\log(x + 1) + \log(x^2 + 1)$	35%
Nguyen-8	\sqrt{x}	96%
Nguyen-9	$\sin(x) + \sin(y^2)$	100%
Nguyen-10	$2 \sin(x) \cos(y)$	100%
Nguyen-11	x^y	100%
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	0%
	Average	83.6%

Performance of DSR with Noisy data

Add Gaussian noise to y , where the noise has mean 0 and standard deviation proportional to the root-mean-square error of y .

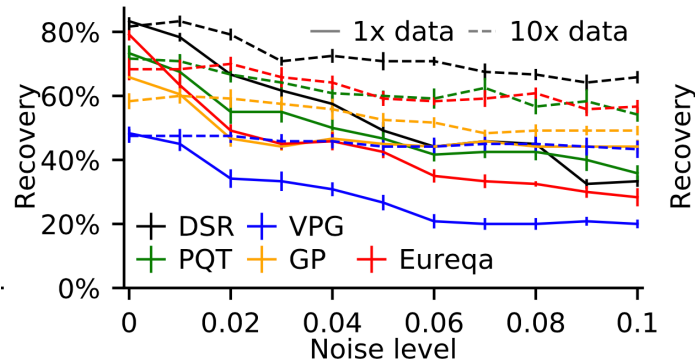


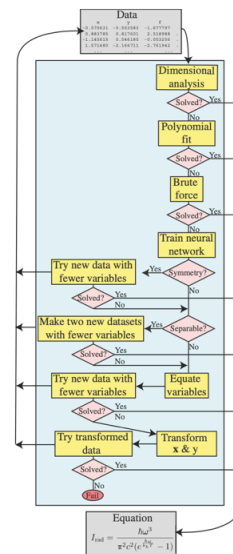
Figure 4: Recovery vs dataset noise and dataset size across all Nguyen benchmarks. Error bars represent standard error.

Comparison of Running Time

Without early stopping, Genetic Programming is faster, as it does not require neural network training.

Considering early stopping, Deep Symbolic Regression can be faster when it has a high recovery rate and as a result can trigger early stopping more often.

Equation Simplification for Symbolic Regression



Equation Simplification for Symbolic Regression

Name of the algorithm: **AI Feynman**

Benchmark equations: 100 equations from “Feynman Lectures on Physics”, which have 1 to 9 independent variables.
Elementary functions: +, -, *, /, sqrt, exp, log, sin, cos, arcsin, tanh.

Feynman Eq.	Equation
l.6.20a	$f = e^{-\theta^2/2}/\sqrt{2\pi}$
l.6.20	$f = e^{-\frac{\theta^2}{2\sigma^2}}/\sqrt{2\pi\sigma^2}$
l.6.20b	$f = e^{-\frac{(\theta-\theta_0)^2}{2\sigma^2}}/\sqrt{2\pi\sigma^2}$
l.8.14	$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
l.9.18	$F = \frac{Gm_1m_2}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$
l.10.7	$m = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}}$
l.11.19	$A = x_1y_1 + x_2y_2 + x_3y_3$
l.12.1	$F = \mu N_n$

l.12.2	$F = \frac{q_1q_2}{4\pi\epsilon r^2}$
l.12.4	$E_f = \frac{q_1}{4\pi\epsilon r^2}$
l.12.5	$F = q_2E_f$
l.12.11	$F = q(E_f + Bv \sin \theta)$
l.13.4	$K = \frac{1}{2}m(v^2 + u^2 + w^2)$
l.13.12	$U = Gm_1m_2(\frac{1}{r_2} - \frac{1}{r_1})$
l.14.3	$U = mgz$
l.14.4	$U = \frac{k_{spring}x^2}{2}$
l.15.3x	$x_1 = \frac{x - ut}{\sqrt{1 - u^2/c^2}}$
l.15.3t	$t_1 = \frac{t - ux/c^2}{\sqrt{1 - u^2/c^2}}$

l.15.10	$p = \frac{m_0v}{\sqrt{1 - v^2/c^2}}$
l.16.6	$v_1 = \frac{u + v}{1 + uv/c^2}$
l.18.4	$r = \frac{m_1r_1 + m_2r_2}{m_1 + m_2}$
l.18.12	$\tau = rF \sin \theta$
l.18.16	$L = mrv \sin \theta$
l.24.6	$E = \frac{1}{4}m(\omega^2 + \omega_0^2)x^2$
l.25.13	$V_e = \frac{q}{C}$

Feynman Eq.	Equation
I.26.2	$\theta_1 = \arcsin(n \sin \theta_2)$
I.27.6	$f_f = \frac{1}{\frac{1}{d_1} + \frac{n}{d_2}}$
I.29.4	$k = \frac{\omega}{c}$
I.29.16	$x = \sqrt{x_1^2 + x_2^2 - 2x_1x_2 \cos(\theta_1 - \theta_2)}$
I.30.3	$I_* = I_*^0 \frac{\sin^2(n\theta/2)}{\sin^2(\theta/2)}$
I.30.5	$\theta = \arcsin\left(\frac{\lambda}{nd}\right)$
I.32.5	$P = \frac{q^2 a^2}{6\pi\epsilon c^3}$
I.32.17	$P = \left(\frac{1}{2}\epsilon c E_f^2\right) \left(\frac{8\pi r^2}{3}\right) \frac{\omega^4}{(\omega^2 - \omega_0^2)^2}$

I.34.8	$\omega = \frac{qvB}{p}$
I.34.10	$\omega = \frac{\omega_0}{1 - v/c}$
I.34.14	$\omega = \frac{1 + v/c}{\sqrt{1 - v^2/c^2}} \omega_0$
I.34.27	$E = \hbar\omega$
I.37.4	$I_* = I_1 + I_2 + 2\sqrt{I_1 I_2} \cos \delta$
I.38.12	$r = \frac{4\pi\epsilon \hbar^2}{mq^2}$
I.39.10	$E = \frac{3}{2} p_F V$
I.39.11	$E = \frac{1}{\gamma - 1} p_F V$
I.39.22	$P_F = \frac{nk_B T}{V}$

I.40.1	$n = n_0 e^{-\frac{mgx}{k_B T}}$
I.41.16	$L_{\text{rad}} = \frac{\hbar \omega^3}{\pi^2 c^2 (e^{\frac{\hbar \omega}{k_B T}} - 1)}$
I.43.16	$v = \frac{\mu_{\text{drift}} q V_e}{d}$
I.43.31	$D = \mu_e k_B T$
I.43.43	$\kappa = \frac{1}{\gamma - 1} \frac{k_B v}{A}$
I.44.4	$E = nk_B T \ln\left(\frac{V_2}{V_1}\right)$
I.47.23	$c = \sqrt{\frac{\gamma p r}{\rho}}$
I.48.20	$E = \frac{mc^2}{\sqrt{1 - v^2/c^2}}$
I.50.26	$x = x_1 [\cos(\omega t) + \alpha \cos(\omega t)^2]$

Feynman Eq.	Equation
II.2.42	$P = \frac{\kappa(T_2 - T_1)A}{d}$
II.3.24	$F_E = \frac{P}{4\pi r^2}$
II.4.23	$V_e = \frac{q}{4\pi\epsilon r}$
II.6.11	$V_e = \frac{1}{4\pi\epsilon} \frac{p_d \cos \theta}{r^2}$
II.6.15a	$E_f = \frac{3}{4\pi\epsilon} \frac{p_d z}{r^5} \sqrt{x^2 + y^2}$
II.6.15b	$E_f = \frac{3}{4\pi\epsilon} \frac{p_d}{r^3} \cos \theta \sin \theta$
II.8.7	$E = \frac{3}{5} \frac{q^2}{4\pi\epsilon d}$

II.8.31	$E_{\text{den}} = \frac{\epsilon E_f^2}{2}$
II.10.9	$E_f = \frac{\sigma_{\text{den}}}{\epsilon} \frac{1}{1 + \chi}$
II.11.3	$x = \frac{qE_f}{m(\omega_0^2 - \omega^2)}$
II.11.17	$n = n_0 \left(1 + \frac{p_d E_f \cos \theta}{k_b T} \right)$
II.11.20	$P_* = \frac{n_p p_d^2 E_f}{3 k_b T}$
II.11.27	$P_* = \frac{n\alpha}{1 - n\alpha/3} \epsilon E_f$
II.11.28	$\theta = 1 + \frac{n\alpha}{1 - (n\alpha/3)}$
II.13.17	$B = \frac{1}{4\pi\epsilon c^2} \frac{2I}{r}$

II.13.23	$\rho_c = \frac{\rho_{c_0}}{\sqrt{1 - v^2/c^2}}$
II.13.34	$j = \frac{\rho_{c_0} v}{\sqrt{1 - v^2/c^2}}$
II.15.4	$E = -\mu_M B \cos \theta$
II.15.5	$E = -\rho_d E_f \cos \theta$
II.21.32	$V_e = \frac{q}{4\pi\epsilon r(1 - v/c)}$
II.24.17	$k = \sqrt{\frac{\omega^2}{c^2} - \frac{\pi^2}{d^2}}$
II.27.16	$F_E = \epsilon c E_f^2$
II.27.18	$E_{\text{den}} = \epsilon E_f^2$

Feynman Eq.	Equation
II.34.2a	$I = \frac{qV}{2\pi r}$
II.34.2	$\mu_M = \frac{qvr}{2}$
II.34.11	$\omega = \frac{g \cdot qB}{2m}$
II.34.29a	$\mu_M = \frac{qh}{4\pi m}$
II.34.29b	$E = \frac{g \cdot \mu_M B J_z}{\hbar}$
II.35.18	$n = \frac{n_0}{\exp(\mu_m B / (k_b T)) + \exp(-\mu_m B / (k_b T))}$
II.35.21	$M = n_p \mu_M \tanh\left(\frac{\mu_M B}{k_b T}\right)$

II.36.38	$f = \frac{\mu_m B}{k_b T} + \frac{\mu_m \alpha M}{\epsilon c^2 k_b T}$
II.37.1	$E = \mu_M (1 + \chi) B$
II.38.3	$F = \frac{YAx}{d}$
II.38.14	$\mu_s = \frac{Y}{2(1 + \sigma)}$
III.4.32	$n = \frac{1}{e^{\frac{\hbar \omega}{k_b T}} - 1}$
III.4.33	$E = \frac{\hbar \omega}{e^{\frac{\hbar \omega}{k_b T}} - 1}$
III.7.38	$\omega = \frac{2\mu_M B}{\hbar}$
III.8.54	$p_Y = \sin\left(\frac{Et}{\hbar}\right)^2$
III.9.52	$p_Y = \frac{p_d E t \sin((\omega - \omega_0) t / 2)^2}{\hbar ((\omega - \omega_0) t / 2)^2}$
III.10.19	$E = \mu_M \sqrt{B_x^2 + B_y^2 + B_z^2}$

III.12.43	$L = n\hbar$
III.13.18	$v = \frac{2Ed^2 k}{\hbar}$
III.14.14	$I = I_0 (e^{\frac{qV_e}{k_b T}} - 1)$
III.15.12	$E = 2U(1 - \cos(kd))$
III.15.14	$m = \frac{\hbar^2}{2Ed^2}$
III.15.27	$k = \frac{2\pi\alpha}{nd}$
III.17.37	$f = \beta(1 + \alpha \cos \theta)$
III.19.51	$E = \frac{-mq^4}{2(4\pi\epsilon)^2 \hbar^2 n^2} \frac{1}{n^2}$
III.21.20	$j = \frac{-\rho_{c_0} q A_{vec}}{m}$

Bonus equations for testing

Source	Equation
Rutherford scattering	$A = \left(\frac{Z_1 Z_2 \alpha \hbar c}{4E \sin^2(\frac{\theta}{2})} \right)^2$
Friedman equation	$H = \sqrt{\frac{8\pi G}{3} \rho - \frac{k_f c^2}{a_f^2}}$
Compton scattering	$U = \frac{E}{1 + \frac{E}{m_e c^2} (1 - \cos \theta)}$
Radiated gravitational wave power	$P = -\frac{32 G^4 (m_1 m_2)^2 (m_1 + m_2)}{5 c^5 r^5}$
Relativistic aberration	$\theta_1 = \arccos \left(\frac{\cos \theta_2 - \frac{v}{c}}{1 - \frac{v}{c} \cos \theta_2} \right)$
N-slit diffraction	$I = I_0 \left[\frac{\sin(\alpha/2)}{\alpha/2} \frac{\sin(N\delta/2)}{\sin(\delta/2)} \right]^2$
Goldstein 3.16	$v = \sqrt{\frac{2}{m} \left(E - U - \frac{L^2}{2mr^2} \right)}$

Goldstein 3.55

$$k = \frac{mk_G}{L^2} \left(1 + \sqrt{1 + \frac{2EL^2}{mk_G^2} \cos(\theta_1 - \theta_2)} \right)$$

Goldstein 3.64 (ellipse)

$$r = \frac{d(1 - \alpha^2)}{1 + \alpha \cos(\theta_1 - \theta_2)}$$

Goldstein 3.74 (Kepler)

$$t = \frac{2\pi d^{3/2}}{\sqrt{G(m_1 + m_2)}}$$

Goldstein 3.99

$$\alpha = \sqrt{1 + \frac{2e^2 EL^2}{m(Z_1 Z_2 q^2)^2}}$$

Goldstein 8.56

$$E = \sqrt{(p - qA_{\text{vec}})^2 c^2 + m^2 c^4} + qV_e$$

Goldstein 12.80

$$E = \frac{1}{2m} [p^2 + m^2 \omega^2 x^2 (1 + \alpha \frac{x}{y})]$$

Jackson 2.11

$$F = \frac{q}{4\pi\epsilon y^2} \left[4\pi\epsilon V_e d - \frac{qdy^3}{(y^2 - d^2)^2} \right]$$

Bonus equations for testing

Jackson 3.45

$$V_e = \frac{q}{(r^2 + d^2 - 2dr \cos \alpha)^{\frac{3}{2}}}$$

Jackson 4.60

$$V_e = E_f \cos \theta \left(\frac{\alpha - 1}{\alpha + 2} \frac{d^3}{r^2} - r \right)$$

Jackson 11.38 (Doppler)

$$\omega_0 = \frac{\sqrt{1 - \frac{v^2}{c^2}}}{1 + \frac{v}{c} \cos \theta} \omega$$

Weinberg 15.2.1

$$\rho = \frac{3}{8\pi G} \left(\frac{c^2 k_f}{a_f^2} + H^2 \right)$$

Weinberg 15.2.2

$$p_f = -\frac{1}{8\pi G} \left[\frac{c^4 k_f}{a_f^2} + c^2 H^2 (1 - 2\alpha) \right]$$

Schwarz 13.132 (Klein-Nishina)

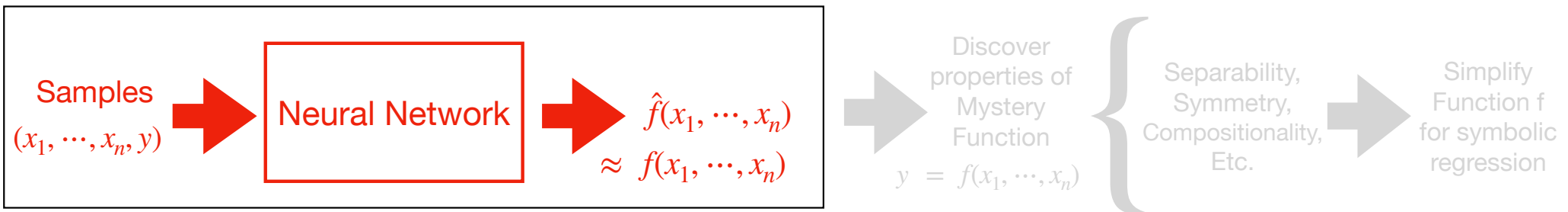
$$A = \frac{\pi \alpha^2 \hbar^2}{m^2 c^2} \left(\frac{\omega_0}{\omega} \right)^2 \left[\frac{\omega_0}{\omega} + \frac{\omega}{\omega_0} - \sin^2 \theta \right]$$

Sampling the Training/Test Equations

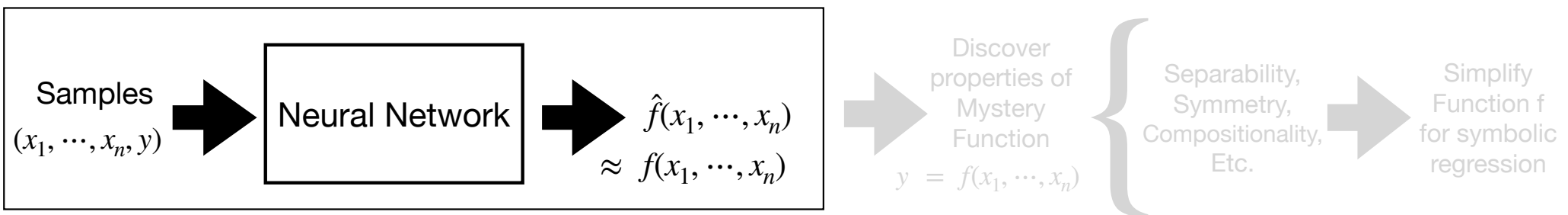
For each equation, 100,000 samples are taken.

Each input variable is uniformly sampled between 1 and 5.

Equation Simplification for Symbolic Regression: **AI Feynman** Algorithm



Equation Simplification for Symbolic Regression: AI Feynman Algorithm



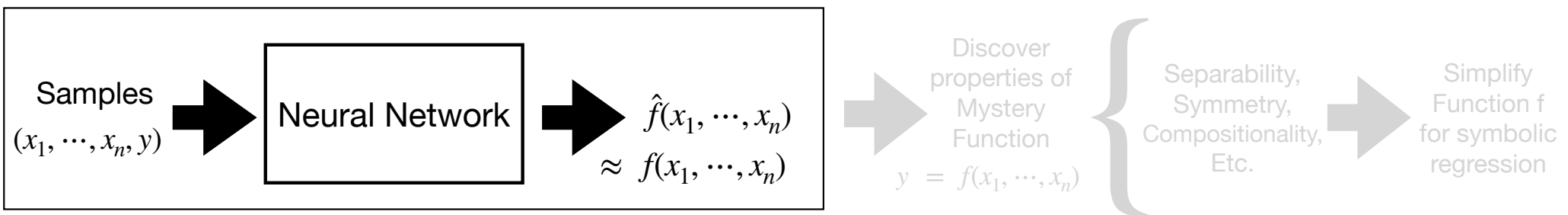
Universal Approximation Theorem

A simpler statement of the “Universal Approximation Theorem” (by Ian Goodfellow):
“A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.”

Multilayer feedforward networks are universal approximations, by Kurt Hornik, Maxwell Tinchcombe, and Halbert White, in *Neural Networks*, vol. 2, pp. 359-366, 1989.

Paper: [AI Feynman: A physics-inspired method for symbolic regression](#), by Silviu-Marian Udrescu and Max Tegmark, in *Science Advances*, 6 (16), eaay: 2631, April 15, 2020.

Equation Simplification for Symbolic Regression: AI Feynman Algorithm

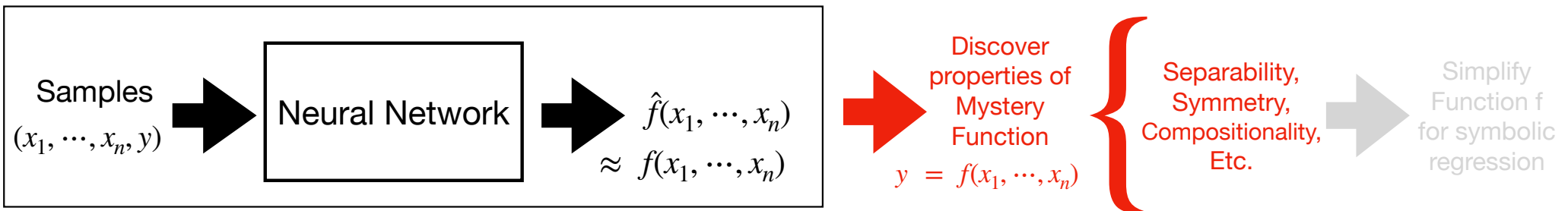


We train a feed-forward, fully connected neural network with six hidden layers with softplus $\log(1 + e^x)$ activation functions, the first three layers having 128 neurons and the last three layers having 64 neurons.

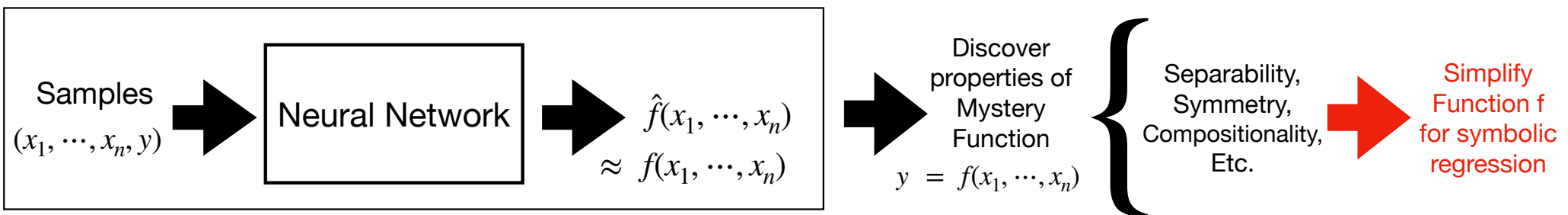
For each mystery function, we generated 100,000 data points, using 80% as the training set and the remainder 20% as the validation set, training for 100 epochs with learning rate 0.005 and batch size 2048.

We use the RMSE loss function and the Adam optimizer with a weight decay of 0.01.

Equation Simplification for Symbolic Regression: AI Feynman Algorithm



Equation Simplification for Symbolic Regression: AI Feynman Algorithm



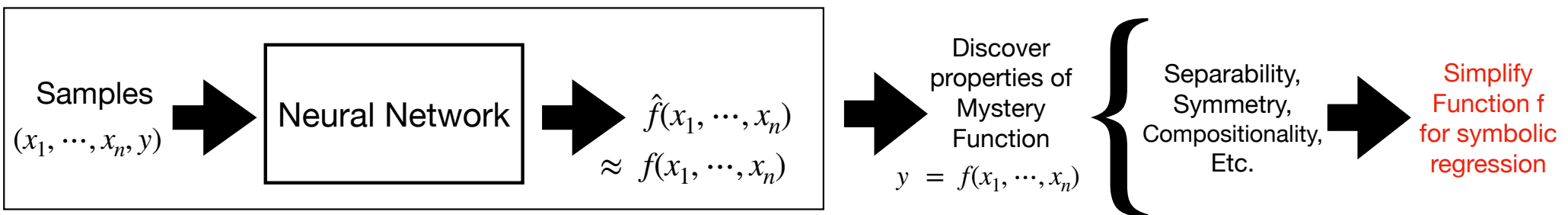
Example: $f = (x_1 - x_2)(x_3^2 + x_3x_4 - x_4 + 3) \rightarrow f_1 = x_1 - x_2$ and $f_2 = x_3^2 + x_3x_4 - x_4 + 3$

Example: $f = (x_1 - x_2)^2 + (x_3 - x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1 - x_2, z_2 = x_3 - x_4$

Example: $f = (x_1/x_2)^2 + (x_3/x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1/x_2, z_2 = x_3/x_4$

Example: $f = x_3\sqrt{(x_1^2 + x_2^2)x_3} \rightarrow f = g(h(x_1, x_2), x_3)$ with $h(x_1, x_2) = x_1^2 + x_2^2$ and $g(h, x_3) = x_3\sqrt{hx_3}$

Equation Simplification for Symbolic Regression: AI Feynman Algorithm



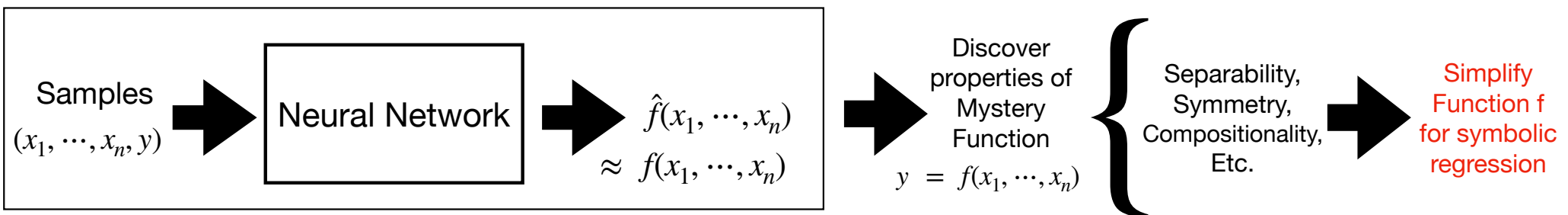
Example: $f = (x_1 - x_2)(x_3^2 + x_3x_4 - x_4 + 3) \rightarrow f_1 = x_1 - x_2$ and $f_2 = x_3^2 + x_3x_4 - x_4 + 3$

Example: $f = (x_1 - x_2)^2 + (x_3 - x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1 - x_2, z_2 = x_3 - x_4$

Example: $f = (x_1/x_2)^2 + (x_3/x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1/x_2, z_2 = x_3/x_4$

Example: $f = x_3\sqrt{(x_1^2 + x_2^2)x_3} \rightarrow f = g(h(x_1, x_2), x_3)$ with $h(x_1, x_2) = x_1^2 + x_2^2$ and $g(h, x_3) = x_3\sqrt{hx_3}$

Equation Simplification for Symbolic Regression: AI Feynman Algorithm



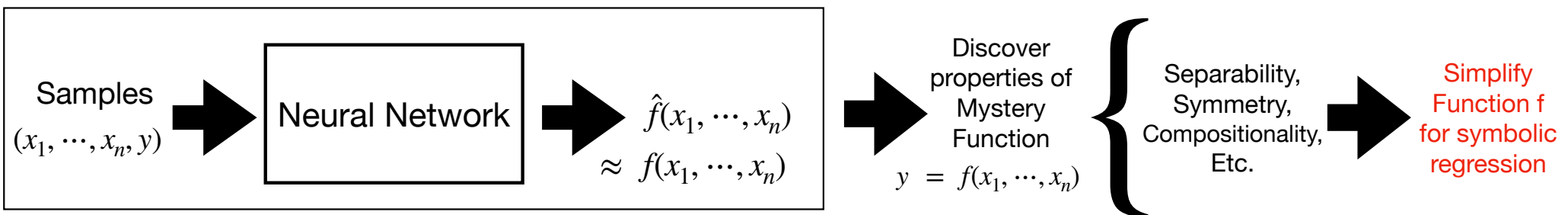
Example: $f = (x_1 - x_2)(x_3^2 + x_3x_4 - x_4 + 3) \rightarrow f_1 = x_1 - x_2$ and $f_2 = x_3^2 + x_3x_4 - x_4 + 3$

Example: $f = (x_1 - x_2)^2 + (x_3 - x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1 - x_2, z_2 = x_3 - x_4$

Example: $f = (x_1/x_2)^2 + (x_3/x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1/x_2, z_2 = x_3/x_4$

Example: $f = x_3\sqrt{(x_1^2 + x_2^2)x_3} \rightarrow f = g(h(x_1, x_2), x_3)$ with $h(x_1, x_2) = x_1^2 + x_2^2$ and $g(h, x_3) = x_3\sqrt{hx_3}$

Equation Simplification for Symbolic Regression: AI Feynman Algorithm



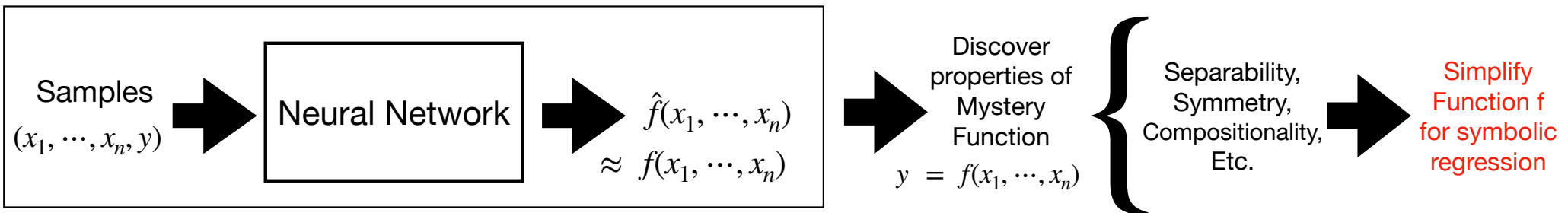
Example: $f = (x_1 - x_2)(x_3^2 + x_3x_4 - x_4 + 3) \rightarrow f_1 = x_1 - x_2$ and $f_2 = x_3^2 + x_3x_4 - x_4 + 3$

Example: $f = (x_1 - x_2)^2 + (x_3 - x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1 - x_2, z_2 = x_3 - x_4$

Example: $f = (x_1/x_2)^2 + (x_3/x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1/x_2, z_2 = x_3/x_4$

Example: $f = x_3\sqrt{(x_1^2 + x_2^2)x_3} \rightarrow f = g(h(x_1, x_2), x_3)$ with $h(x_1, x_2) = x_1^2 + x_2^2$ and $g(h, x_3) = x_3\sqrt{hx_3}$

Equation Simplification for Symbolic Regression: AI Feynman Algorithm



Example: $f = (x_1 - x_2)(x_3^2 + x_3x_4 - x_4 + 3) \rightarrow f_1 = x_1 - x_2$ and $f_2 = x_3^2 + x_3x_4 - x_4 + 3$

Example: $f = (x_1 - x_2)^2 + (x_3 - x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1 - x_2, z_2 = x_3 - x_4$

Example: $f = (x_1/x_2)^2 + (x_3/x_4)^2 \rightarrow f = z_1^2 + z_2^2$ with $z_1 = x_1/x_2, z_2 = x_3/x_4$

Example: $f = x_3\sqrt{(x_1^2 + x_2^2)x_3} \rightarrow f = g(h(x_1, x_2), x_3)$ with $h(x_1, x_2) = x_1^2 + x_2^2$ and $g(h, x_3) = x_3\sqrt{hx_3}$

Functions in practice often have these simplifying properties:

1. **Units:** f and the variables upon which it depends have known physical units.
2. **Low-order polynomial:** f (or part of f) is a polynomial of low degree.
3. **Compositionality:** f is a composition of a small set of elementary functions, each typically taking no more than two arguments.
4. **Smoothness:** f is continuous and perhaps even analytic in its domain.
5. **Symmetry:** f exhibits translational, rotational, or scaling symmetry with respect to some of its variables.
6. **Separability:** f can be written as a sum or product of two parts with no variables in common.

Functions in practice often have these simplifying properties:

1. Units: f and the variables upon which it depends have known physical units.
2. Low-order polynomial: f (or part of f) is a polynomial of low degree.
3. Compositionality: f is a composition of a small set of elementary functions, each typically taking no more than two arguments.
4. Smoothness: f is continuous and perhaps even analytic in its domain.
5. Symmetry: f exhibits translational, rotational, or scaling symmetry with respect to some of its variables.
6. Separability: f can be written as a sum or product of two parts with no variables in common.

Functions in practice often have these simplifying properties:

1. Units: f and the variables upon which it depends have known physical units.
2. Low-order polynomial: f (or part of f) is a polynomial of low degree.
3. Compositionality: f is a composition of a small set of elementary functions, each typically taking no more than two arguments.
4. Smoothness: f is continuous and perhaps even analytic in its domain.
5. Symmetry: f exhibits translational, rotational, or scaling symmetry with respect to some of its variables.
6. Separability: f can be written as a sum or product of two parts with no variables in common.

Functions in practice often have these simplifying properties:

1. Units: f and the variables upon which it depends have known physical units.
2. Low-order polynomial: f (or part of f) is a polynomial of low degree.
3. Compositionality: f is a composition of a small set of elementary functions, each typically taking no more than two arguments.
4. Smoothness: f is continuous and perhaps even analytic in its domain.
5. Symmetry: f exhibits translational, rotational, or scaling symmetry with respect to some of its variables.
6. Separability: f can be written as a sum or product of two parts with no variables in common.

Functions in practice often have these simplifying properties:

1. Units: f and the variables upon which it depends have known physical units.
2. Low-order polynomial: f (or part of f) is a polynomial of low degree.
3. Compositionality: f is a composition of a small set of elementary functions, each typically taking no more than two arguments.
4. Smoothness: f is continuous and perhaps even analytic in its domain.
5. Symmetry: f exhibits translational, rotational, or scaling symmetry with respect to some of its variables.
6. Separability: f can be written as a sum or product of two parts with no variables in common.

Functions in practice often have these simplifying properties:

1. Units: f and the variables upon which it depends have known physical units.
2. Low-order polynomial: f (or part of f) is a polynomial of low degree.
3. Compositionality: f is a composition of a small set of elementary functions, each typically taking no more than two arguments.
4. Smoothness: f is continuous and perhaps even analytic in its domain.
5. Symmetry: f exhibits translational, rotational, or scaling symmetry with respect to some of its variables.
6. Separability: f can be written as a sum or product of two parts with no variables in common.

Functions in practice often have these simplifying properties:

1. Units: f and the variables upon which it depends have known physical units.

Property 1 enables dimensional analysis, which often transforms the problem into a simpler one with fewer independent variables.

Dimensional analysis

Often the problem can be simplified by requiring the units of the two sides of an equation to match.

Fundamental Units: Meter, second, kilogram, kelvin, volt

Table 3. Unit table used for our automated dimensional analysis.

Variables	Units	m	s	kg	T	V
a, g	Acceleration	1	-2	0	0	0
h, \hbar, L, J_2	Angular momentum	2	-1	1	0	0
A	Area	2	0	0	0	0
k_b	Boltzmann constant	2	-2	1	-1	0
C	Capacitance	2	-2	1	0	-2
q, q_1, q_2	Charge	2	-2	1	0	-1
j	Current density	0	-3	1	0	-1
I, I_0	Current Intensity	2	-3	1	0	-1
ρ, ρ_0	Density	-3	0	1	0	0
$\theta, \theta_1, \theta_2, \sigma, n$	Dimensionless	0	0	0	0	0
$g, k_f, \gamma, \chi, \beta, \alpha$	Dimensionless	0	0	0	0	0
$p, \gamma, n_0, \delta, f, \mu$	Dimensionless	0	0	0	0	0
$n_0, \delta, f, \mu, Z_1, Z_2$	Dimensionless	0	0	0	0	0
D	Diffusion coefficient	2	-1	0	0	0
μ_{drift}	Drift velocity constant	0	-1	1	0	0
p_d	Electric dipole moment	3	-2	1	0	-1
E_f	Electric field	-1	0	0	0	1
ϵ	Electric permittivity	1	-2	1	0	-2
E, K, U	Energy	2	-2	1	0	0

Functions in practice often have these simplifying properties:

1. Units: f and the variables upon which it depends have known physical units.
2. Low-order polynomial: f (or part of f) is a polynomial of low degree.

Property 2 enables polynomial fitting, which quickly solves the problem by solving a system of linear equations to determine the polynomial coefficients.

Polynomial Fit

Many functions in practice are low-order polynomials, e.g., the kinetic energy $K = \frac{m}{2}(v_x^2 + v_y^2 + v_z^2)$

or have parts that are, e.g., the denominator of the gravitational force $F = \frac{Gm_1m_2}{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$

We use a module to test if a mystery can be solved by a low-order polynomial.

It uses standard techniques for polynomial fitting, and declares success if the best-fitting polynomial gives root mean square error (RMSE) less than a threshold.

Brute Force Search

When the problem is small enough, simply try all simple (short) symbolic expressions, and declares success when the maximum fitting error is below a threshold.

How to enumerate all simple (short) symbolic expressions: use computation tree, or its string representation.

A string representation: reverse Polish notation

String Representation for Function: Reverse Polish notation

Table 1. Functions optionally included in brute-force search. The following three subsets are tried in turn: "+-*/><~SPLICER", "+-*/> 0~" and "+-*/><~REPLICANTS0".

Symbol	Meaning	Arguments
+	Add	2
*	Multiply	2
-	Subtract	2
/	Divide	2
>	Increment	1
<	Decrement	1
~	Negate	1
0	0	0
1	1	0
R	sqrt	1
E	exp	1
P	π	0
L	ln	1
I	invert	1
C	cos	1
A	abs	1
N	arcsin	1
T	arctan	1
S	sin	1

Examples of "reverse Polish notation" (where parentheses are unnecessary):

$$x + y \rightarrow xy+$$

$$-2/3 \rightarrow 0 < < 1 > > /$$

$$mv/\sqrt{1 - v^2/c^2} \rightarrow mv * 1vv * cc * / - R/$$

The brute-force algorithm solves the constants in the symbolic expressions using non-linear regression, and focuses on enumeration of the remaining part of the symbolic expressions.

String Representation for Function: Reverse Polish notation

Table 1. Functions optionally included in brute-force search. The following three subsets are tried in turn: "+-*/><~SPLICER", "+-*/> 0~" and "+-*/><~REPLICANTS0".

Symbol	Meaning	Arguments
+	Add	2
*	Multiply	2
-	Subtract	2
/	Divide	2
>	Increment	1
<	Decrement	1
~	Negate	1
0	0	0
1	1	0
R	sqrt	1
E	exp	1
P	π	0
L	ln	1
I	invert	1
C	cos	1
A	abs	1
N	arcsin	1
T	arctan	1
S	sin	1

Examples of "reverse Polish notation" (where parentheses are unnecessary):

$$x + y \rightarrow xy+$$

$$-2/3 \rightarrow 0 < < 1 > > /$$

$$mv/\sqrt{1 - v^2/c^2} \rightarrow mv * 1vv * cc * / - R/$$

The brute-force algorithm solves the constants in the symbolic expressions using non-linear regression, and focuses on enumeration of the remaining part of the symbolic expressions.

String Representation for Function: Reverse Polish notation

Table 1. Functions optionally included in brute-force search. The following three subsets are tried in turn: "+*/><~SPLICER", "+*/> 0~" and "+*/><~REPLICANTS0".

Symbol	Meaning	Arguments
+	Add	2
*	Multiply	2
-	Subtract	2
/	Divide	2
>	Increment	1
<	Decrement	1
~	Negate	1
0	0	0
1	1	0
R	sqrt	1
E	exp	1
P	π	0
L	ln	1
I	invert	1
C	cos	1
A	abs	1
N	arcsin	1
T	arctan	1
S	sin	1

Examples of "reverse Polish notation" (where parentheses are unnecessary):

$$x + y \rightarrow xy+$$

$$-2/3 \rightarrow 0 < < 1 > > /$$

$$\begin{array}{r} \underline{-1} \quad \underline{2} \\ \hline -2 \quad 3 \end{array}$$

The brute-force algorithm solves the constants in the symbolic expressions using non-linear regression, and focuses on enumeration of the remaining part of the symbolic expressions.

String Representation for Function: Reverse Polish notation

Table 1. Functions optionally included in brute-force search. The following three subsets are tried in turn: "+-*/><~SPLICER", "+-*/> 0~" and "+-*/><~REPLICANTS0".

Symbol	Meaning	Arguments
+	Add	2
*	Multiply	2
-	Subtract	2
/	Divide	2
>	Increment	1
<	Decrement	1
~	Negate	1
0	0	0
1	1	0
R	sqrt	1
E	exp	1
P	π	0
L	ln	1
I	invert	1
C	cos	1
A	abs	1
N	arcsin	1
T	arctan	1
S	sin	1

Examples of "reverse Polish notation" (where parentheses are unnecessary):

$$x + y \rightarrow xy+$$

$$-2/3 \rightarrow 0 < < 1 > > /$$

$$mv/\sqrt{1 - v^2/c^2} \rightarrow \frac{mv * 1}{\frac{v^2 * c^2}{\sqrt{1 - v^2/c^2}}}$$

String Representation for Function: Reverse Polish notation

Table 1. Functions optionally included in brute-force search. The following three subsets are tried in turn: “+*/><~SPLICER”, “+*/> 0~” and “+*/><~REPLICANTS0”.

Symbol	Meaning	Arguments
+	Add	2
*	Multiply	2
-	Subtract	2
/	Divide	2
>	Increment	1
<	Decrement	1
~	Negate	1
0	0	0
1	1	0
R	sqrt	1
E	exp	1
P	π	0
L	ln	1
I	invert	1
C	cos	1
A	abs	1
N	arcsin	1
T	arctan	1
S	sin	1

Examples of “reverse Polish notation” (where parentheses are unnecessary):

$$x + y \rightarrow xy+$$

$$-2/3 \rightarrow 0 < < 1 > > /$$

$$mv\sqrt{1 - v^2/c^2} \rightarrow mv * 1vv * cc * / - R/$$

The brute-force algorithm solves the constants in the symbolic expressions using non-linear regression, and focuses on enumeration of the remaining part of the symbolic expressions.

Balance Accuracy & Complexity

Define the winning function to be the one with RMSE (root mean square error) less than a threshold that has the smallest total description length:

$$DL = \log_2 N + \lambda \log_2 \left[\max\left(1, \frac{\epsilon}{\epsilon_d}\right) \right]$$

where $\epsilon_d = 10^{-15}$ and N is the rank of the string on the list of all strings tried.

The two terms correspond roughly to the number of bits required to store the symbol string and the prediction errors, if the hyperparameter λ is set to equal the number of data points N_d .

In experiments, we use $\lambda = \sqrt{N_d}$ to prioritize simpler formulas.

Test “Translational Symmetry and Generalizations”

We test for translational symmetry using the trained neural network (that approximates the mystery function f).

We first check if $f(x_1, x_2, x_3, \dots) = f(x_1 + a, x_2 + a, x_3, \dots)$ to within a precision threshold.

If that is the case, then we replace x_1 and x_2 by $x'_1 = x_2 - x_1$.

Otherwise, we repeat this test for all pairs of input variables, and also test whether any variable pair can be replaced by its sum, product, or ratio.

If any of these simplifying properties is found, the resulting transformed mystery (with one fewer input variable) is interactively passed into a fresh instantiation of the full AI Feynman symbolic regression algorithm.

Test “Translational Symmetry and Generalizations”

We test for translational symmetry using the trained neural network (that approximates the mystery function f).

We first check if $f(x_1, x_2, x_3, \dots) = f(x_1 + a, x_2 + a, x_3, \dots)$ to within a precision threshold.

If that is the case, then we replace x_1 and x_2 by $x'_1 = x_2 - x_1$.

Otherwise, we repeat this test for all pairs of input variables, and also test whether any variable pair can be replaced by its sum, product, or ratio.

If any of these simplifying properties is found, the resulting transformed mystery (with one fewer input variable) is interactively passed into a fresh instantiation of the full AI Feynman symbolic regression algorithm.

Test “Translational Symmetry and Generalizations”

We test for translational symmetry using the trained neural network (that approximates the mystery function f).

We first check if $f(x_1, x_2, x_3, \dots) = f(x_1 + a, x_2 + a, x_3, \dots)$ to within a precision threshold.

If that is the case, then we replace x_1 and x_2 by $x'_1 = x_2 - x_1$.

Otherwise, we repeat this test for all pairs of input variables, and also test whether any variable pair can be replaced by its sum, product, or ratio.

If any of these simplifying properties is found, the resulting transformed mystery (with one fewer input variable) is interactively passed into a fresh instantiation of the full AI Feynman symbolic regression algorithm.

Test “Separability”

We test for “Separability” using the trained neural network.

A function is separable if it can be split into two parts with no variables in common.

We test for both “Additive Separability” and “Multiplicative Separability”, corresponding to these two parts being added and multiplied, respectively.

For example, to test whether a function of two variables is multiplicatively separable, i.e., of the form

$$f(x_1, x_2) = g(x_1)h(x_2)$$

for some univariate functions g and h , we first select two constants c_1 and c_2 ; for numerical robustness, we choose $c_{\{l\}}$ to be the means of all the values of $x_{\{l\}}$ in the mystery dataset, for $l=1,2$. We then compute the quantity

$$\Delta_{sep}(x_1, x_2) = \frac{1}{f_{rms}} \cdot \left| f(x_1, x_2) - \frac{f(x_1, c_2)f(c_1, x_2)}{f(c_1, c_2)} \right|$$

For each data point. This is a measure of non-separability, since it vanishes if f is multiplicatively separable.

Test “Separability”

We test for “Separability” using the trained neural network.

A function is separable if it can be split into two parts with no variables in common.

We test for both “Additive Separability” and “Multiplicative Separability”, corresponding to these two parts being added and multiplied, respectively.

For example, to test whether a function of two variables is multiplicatively separable, i.e., of the form

$$f(x_1, x_2) = g(x_1)h(x_2)$$

for some univariate functions g and h , we first select two constants c_1 and c_2 ; for numerical robustness, we choose $c_{\{l\}}$ to be the means of all the values of $x_{\{l\}}$ in the mystery dataset, for $l=1,2$. We then compute the quantity

$$\Delta_{sep}(x_1, x_2) = \frac{1}{f_{rms}} \cdot \left| f(x_1, x_2) - \frac{f(x_1, c_2)f(c_1, x_2)}{f(c_1, c_2)} \right|$$

For each data point. This is a measure of non-separability, since it vanishes if f is multiplicatively separable.

Test “Separability”

We test for “Separability” using the trained neural network.

A function is separable if it can be split into two parts with no variables in common.

We test for both “Additive Separability” and “Multiplicative Separability”, corresponding to these two parts being added and multiplied, respectively.

For example, to test whether a function of two variables is multiplicatively separable, i.e., of the form

$$f(x_1, x_2) = g(x_1)h(x_2)$$

for some univariate functions g and h , we first select two constants c_1 and c_2 ; for numerical robustness, we choose $c_{\{i\}}$ to be the means of all the values of $x_{\{i\}}$ in the mystery dataset, for $i=1,2$. We then compute the quantity

$$\Delta_{sep}(x_1, x_2) = \frac{1}{f_{rms}} \cdot \left| f(x_1, x_2) - \frac{f(x_1, c_2)f(c_1, x_2)}{f(c_1, c_2)} \right|$$

for each data point. This is a measure of non-separability, since it vanishes if f is multiplicatively separable.

Use “Separability” to simplify symbolic regression

If separability is found, we define the two univariate mysteries $y' = f(x_1, c_2)$ and $y'' = f(c_1, x_2)/f(c_1, c_2)$.

We pass the first one, $y' = f(x_1, c_2)$ back to fresh instantiation of our full AI Feynman symbolic regression algorithm, and if it gets solved, we redefine $y'' = \frac{y}{y'} \cdot c_{num}$, where c_{num} represents any multiplicative numerical constant that appears in y' .

We then pass y'' back to our algorithm, and if it gets solved, the final solution is $y = \frac{y'y''}{c_{num}}$.

We test for “additive separability” analogously, simply replacing $*$ and $/$ by $+$ and $-$ above; also, c_{num} will represent an additive numerical constant in this case. If we succeed in solving the two parts, then the full solution to the original mystery is the sum of the two parts minus the numerical constant.

When there are more than two variables, we test all the possible subsets of variables that can lead to separability and proceed as above for the newly created two mysteries.

Paper: [AI Feynman: A physics-inspired method for symbolic regression](#), by Silviu-Marian Udrescu and Max Tegmark, in Science Advances, 6 (16), eaay: 2631, April 15, 2020.

Use “Separability” to simplify symbolic regression

If separability is found, we define the two univariate mysteries $y' = f(x_1, c_2)$ and $y'' = f(c_1, x_2)/f(c_1, c_2)$.

We pass the first one, $y' = f(x_1, c_2)$ back to fresh instantiation of our full AI Feynman symbolic regression algorithm, and if it gets solved, we redefine $y'' = \frac{y}{y'} \cdot c_{num}$, where c_{num} represents any multiplicative numerical constant that appears in y' .

We then pass y'' back to our algorithm, and if it gets solved, the final solution is $y = \frac{y'y''}{c_{num}}$.

We test for “additive separability” analogously, simply replacing $*$ and $/$ by $+$ and $-$ above; also, c_{num} will represent an additive numerical constant in this case. If we succeed in solving the two parts, then the full solution to the original mystery is the sum of the two parts minus the numerical constant.

When there are more than two variables, we test all the possible subsets of variables that can lead to separability and proceed as above for the newly created two mysteries.

Paper: [AI Feynman: A physics-inspired method for symbolic regression](#), by Silviu-Marian Udrescu and Max Tegmark, in Science Advances, 6 (16), eaay: 2631, April 15, 2020.

Use “Separability” to simplify symbolic regression

If separability is found, we define the two univariate mysteries $y' = f(x_1, c_2)$ and $y'' = f(c_1, x_2)/f(c_1, c_2)$.

We pass the first one, $y' = f(x_1, c_2)$ back to fresh instantiation of our full AI Feynman symbolic regression algorithm, and if it gets solved, we redefine $y'' = \frac{y}{y'} \cdot c_{num}$, where c_{num} represents any multiplicative numerical constant that appears in y' .

We then pass y'' back to our algorithm, and if it gets solved, the final solution is $y = \frac{y'y''}{c_{num}}$.

We test for “additive separability” analogously, simply replacing $*$ and $/$ by $+$ and $-$ above; also, c_{num} will represent an additive numerical constant in this case. If we succeed in solving the two parts, then the full solution to the original mystery is the sum of the two parts minus the numerical constant.

When there are more than two variables, we test all the possible subsets of variables that can lead to separability and proceed as above for the newly created two mysteries.

Paper: [AI Feynman: A physics-inspired method for symbolic regression](#), by Silviu-Marian Udrescu and Max Tegmark, in Science Advances, 6 (16), eaay: 2631, April 15, 2020.

Use “Separability” to simplify symbolic regression

If separability is found, we define the two univariate mysteries $y' = f(x_1, c_2)$ and $y'' = f(c_1, x_2)/f(c_1, c_2)$.

We pass the first one, $y' = f(x_1, c_2)$ back to fresh instantiation of our full AI Feynman symbolic regression algorithm, and if it gets solved, we redefine $y'' = \frac{y}{y'} \cdot c_{num}$, where c_{num} represents any multiplicative numerical constant that appears in y' .

We then pass y'' back to our algorithm, and if it gets solved, the final solution is $y = \frac{y'y''}{c_{num}}$.

We test for “additive separability” analogously, simply replacing * and / by + and - above; also, c_{num} will represent an additive numerical constant in this case. If we succeed in solving the two parts, then the full solution to the original mystery is the sum of the two parts minus the numerical constant.

When there are more than two variables, we test all the possible subsets of variables that can lead to separability and proceed as above for the newly created two mysteries.

Paper: [AI Feynman: A physics-inspired method for symbolic regression](#), by Silviu-Marian Udrescu and Max Tegmark, in Science Advances, 6 (16), eaay: 2631, April 15, 2020.

Use “Separability” to simplify symbolic regression

If separability is found, we define the two univariate mysteries $y' = f(x_1, c_2)$ and $y'' = f(c_1, x_2)/f(c_1, c_2)$.

We pass the first one, $y' = f(x_1, c_2)$ back to fresh instantiation of our full AI Feynman symbolic regression algorithm, and if it gets solved, we redefine $y'' = \frac{y}{y'} \cdot c_{num}$, where c_{num} represents any multiplicative numerical constant that appears in y' .

We then pass y'' back to our algorithm, and if it gets solved, the final solution is $y = \frac{y'y''}{c_{num}}$.

We test for “additive separability” analogously, simply replacing $*$ and $/$ by $+$ and $-$ above; also, c_{num} will represent an additive numerical constant in this case. If we succeed in solving the two parts, then the full solution to the original mystery is the sum of the two parts minus the numerical constant.

When there are more than two variables, we test all the possible subsets of variables that can lead to separability and proceed as above for the newly created two mysteries.

Extra transformations

Apply the following transformations to variables and y , and try to solve the mystery equation: square root, raise to the power of 2, log, exp, inverse, sin, cos, tan, arcsin, arccos, and arctan.

Example: it may be hard to discover the symbolic equation for $r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$

but if we transform r to r^2 , we can use polynomial regression to solve

$$r^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

Compare the “AI Feynman Algorithm” to Genetic Programming

AI Feynman is good at finding complex equations compared to Genetic Programming.

For example, the neural network strategy is used six times when solving

$$F = \frac{Gm_1m_2}{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Without dimensional analysis: three times to discover translational symmetry that replaces

$$x_1 - x_2, y_1 - y_2, z_1 - z_2,$$

then use separability to repeatedly simplify the function.

An example of how a particular mystery dataset (Newton's law of gravitation with nine variables) is solved.

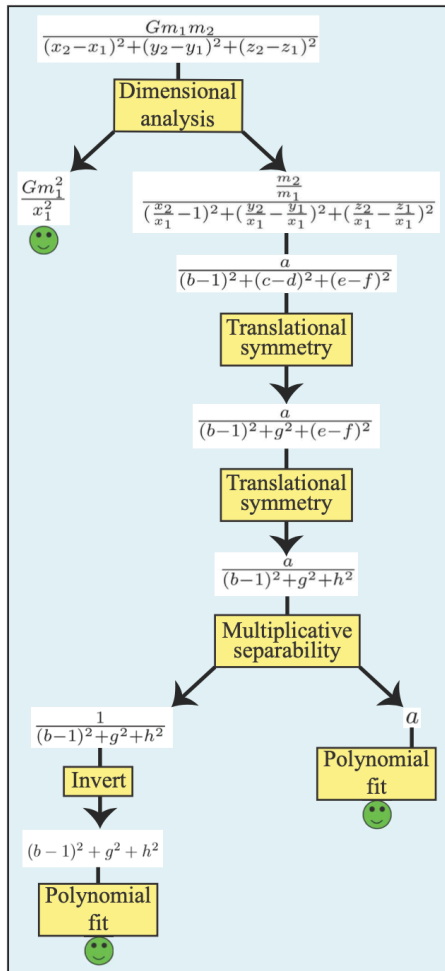


Fig. 2. Example: How our AI Feynman algorithm discovered mystery Equation 5. Given a mystery table with many examples of the gravitational force F together with the nine independent variables $G, m_1, m_2, x_1, \dots, z_2$, this table was recursively transformed into simpler ones until the correct equation was found. First, dimensional analysis generated a table of six dimensionless independent variables $a = m_2/m_1, \dots, f = z_1/x_1$ and the dimensionless dependent variable $\mathcal{F} \equiv F \div Gm_1^2/x_1^2$. Then, a neural network was trained to fit this function, which revealed two translational symmetries (each eliminating one variable, by defining $g \equiv c-d$ and $h \equiv e-f$) as well as multiplicative separability, enabling the factorization $\mathcal{F}(a, b, g, h) = G(a)H(b, g, h)$, thus splitting the problem into two simpler ones. Both G and H then were solved by polynomial fitting, the latter after applying one of a series of simple transformations (in this case, inversion). For many other mysteries, the final step was instead solved using brute-force symbolic search as described in the text.

AI Feynman

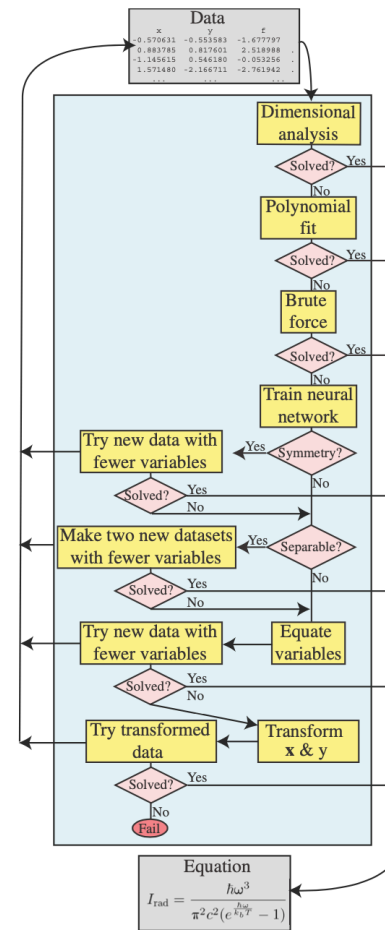
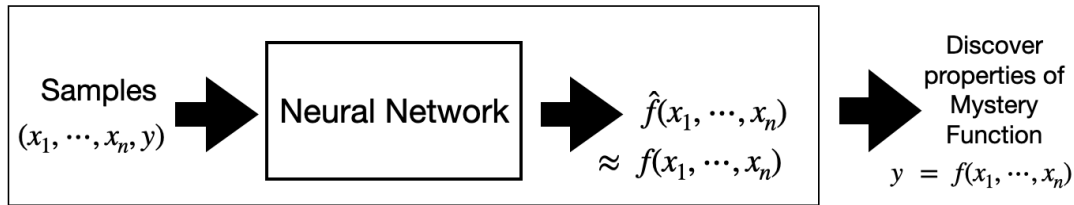


Fig. 1. Schematic illustration of our AI Feynman algorithm. It is iterative as described in the text, with four of the steps capable of generating new mystery datasets that get sent to fresh instantiations of the algorithm, which may or may not return a solution.

Find Pareto-optimal functions: A more robust approach

A more robust approach: find formulas that are Pareto-optimal, in the sense of having the best accuracy for a given complexity. It can make the symbolic regression algorithm orders of magnitude more robust toward noise and bad data.

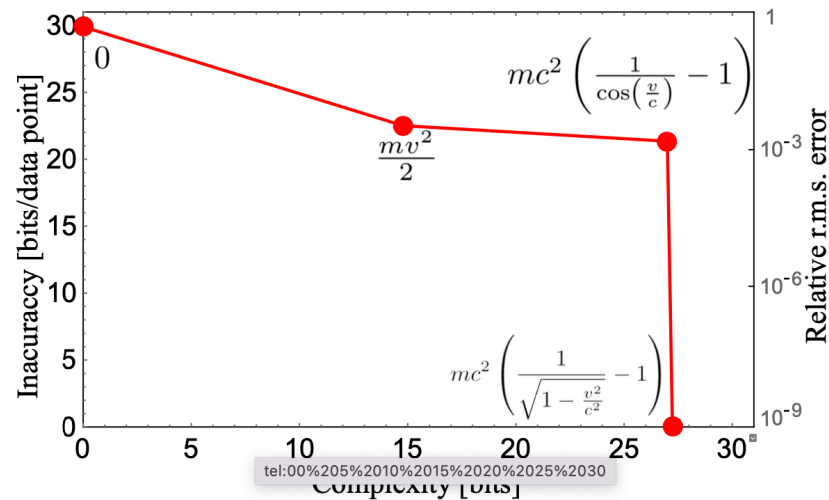


Figure 1: Our symbolic regression of data on how kinetic energy depends on mass, velocity and the speed of light discovers a Pareto-frontier of four formulas that are each the most accurate given their complexity. Convex corners reveal particularly useful formulas, in this case Einstein's formula and the classical approximation $mv^2/2$.

Use more general modality of computation graph to simplify function

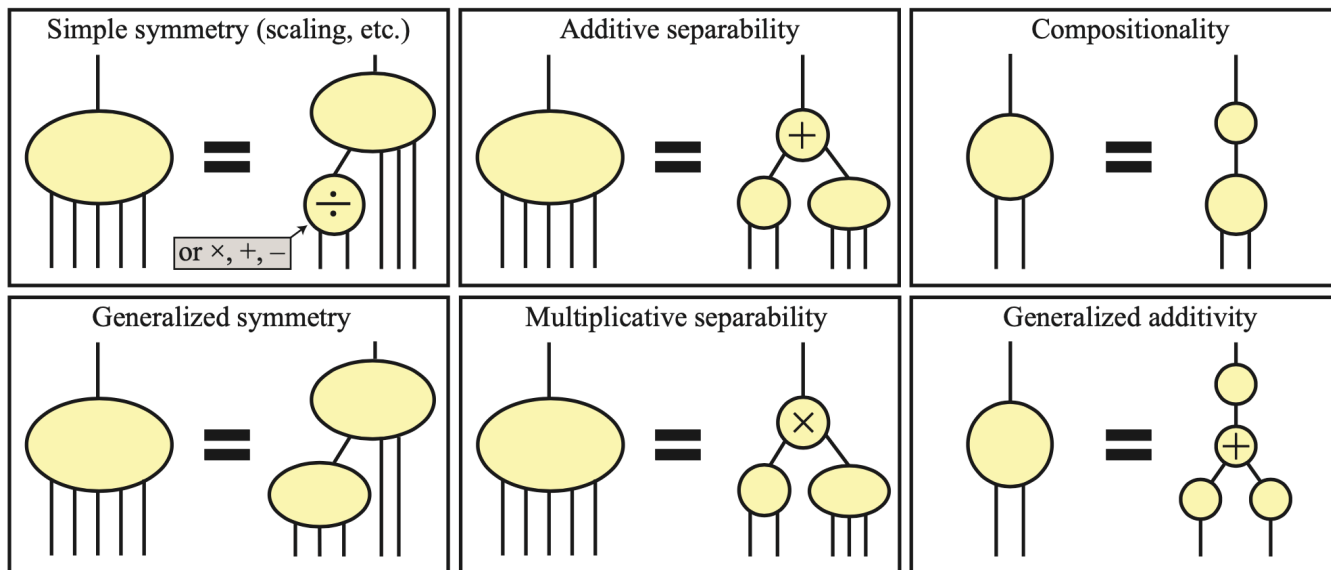


Figure 3: Examples of graph modularity that our algorithm can auto-discover. Lines denote real-valued variables and ovals denote functions, with larger ones being more complex.

An example of generalized symmetry

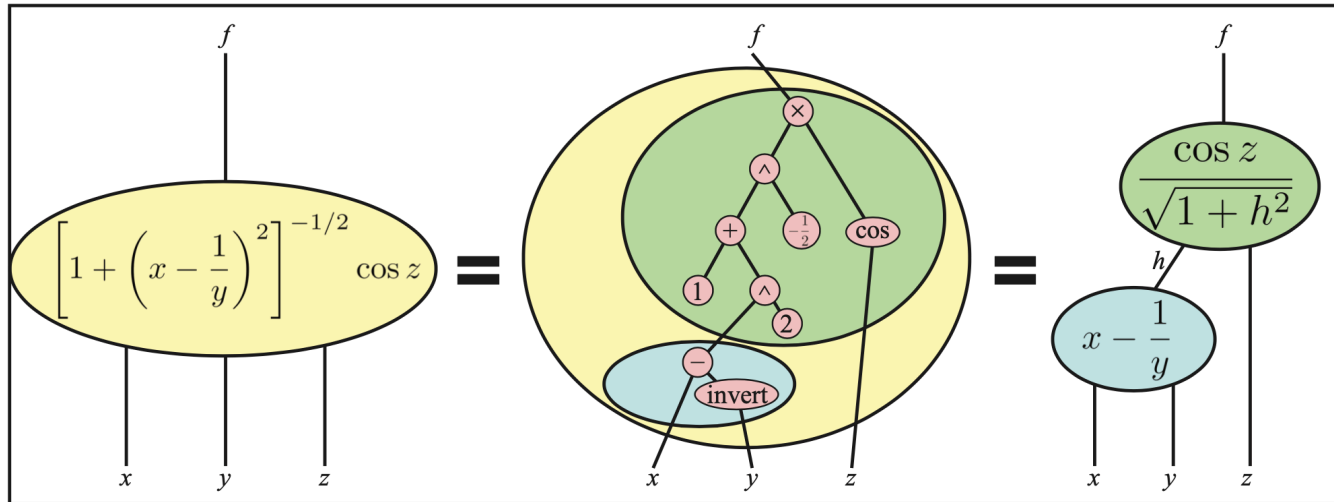
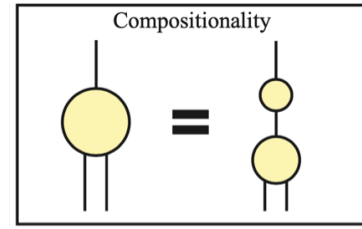


Figure 2: All functions can be represented as tree graphs whose nodes represent a set of basic functions (middle panel). Using a neural network trained to fit a mystery function (left panel), our algorithm seeks a decomposition of this function into others with fewer input variables (right panel), in this case of the form $f(x, y, z) = g[h(x, y), z]$.

Test “Compositionality”

Compositionality: Let us first consider the case of “compositionally”, where $f(\mathbf{x}) = g(h(\mathbf{x}))$ and h is a scalar function simpler than f in the sense of being expressible with a smaller graph.



By the chain rule, we have $\nabla f(\mathbf{x}) = g'(h(\mathbf{x})) \nabla h(\mathbf{x})$

so $\widehat{\nabla} f = \pm \widehat{\nabla} h$ where hats denote unit vectors: $\widehat{\nabla} f = \frac{\nabla f}{|\nabla f|}$ $\widehat{\nabla} h = \frac{\nabla h}{|\nabla h|}$

This means that if we can discover a function h whose gradient is proportional to that of f (for details see the reference paper below), then we can simply replace the variables x in the original mystery data by the single variable $h(x)$ and recursively apply the AI Feynman algorithm to the new one-dimensional symbolic regression problem of discovering $g(h)$.

Test “Generalized Symmetry”

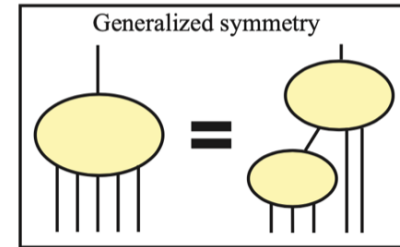
Generalized symmetry: Let us now turn to “Generalized Symmetry”, where k of the n arguments enter only via some scalar function h of them. Specifically, we say that an f has “Generalized Symmetry” if the n components of the vector $\mathbf{x} \in R^n$ can be split into groups of k and $n-k$ components (which we denote by the vectors

$$\mathbf{x}' \in R^k \quad \text{and} \quad \mathbf{x}'' \in R^{n-k}$$

such that $f(\mathbf{x}) = f(\mathbf{x}', \mathbf{x}'') = g[h(\mathbf{x}'), \mathbf{x}'']$ for some function g).

By the chain rule, we have $\nabla_{\mathbf{x}'} f(\mathbf{x}', \mathbf{x}'') = g_1[h(\mathbf{x}'), \mathbf{x}''] \nabla h(\mathbf{x}')$

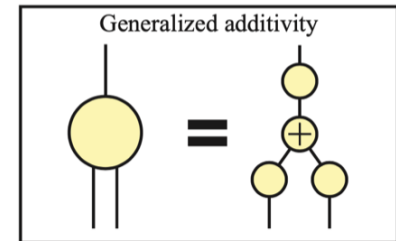
So $\widehat{\nabla_{\mathbf{x}'} f} = \pm \widehat{\nabla h}$ where g_1 denotes the derivative of g with respect to its first argument.



Test “Generalized Additivity”

Generalized Additivity: If f is a function of two variables, then we also test for “Generalized Additivity” where $f(x_1, x_2) = F[g(x_1) + h(x_2)]$

If we define the function $s(x_1, x_2) = \frac{\partial f / \partial x_1}{\partial f / \partial x_2}$ then $s(x_1, x_2) = \frac{g'(x_1)}{h'(x_2)}$



if f satisfies the generalized additivity property. In other words, we simply need to test if s is of the “multiplicatively separable form”

$$s(x_1, x_2) = a(x_1)b(x_2)$$

which we already know how to test.

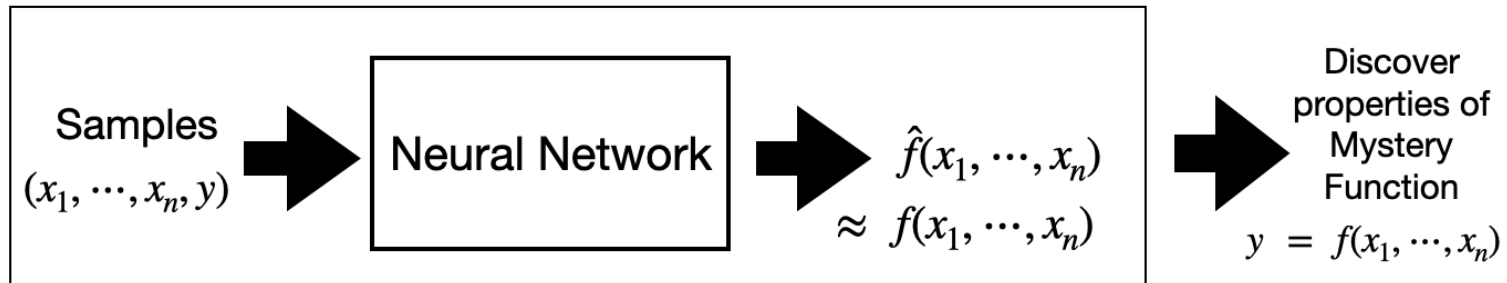
Table 1: Simplification strategies

Name	Property	Action
Negativity	$f(x_1, x_2, \dots) < 0$	Solve for $g \equiv -f$
Positivity	$f(x_1, x_2, \dots) > 0$	Solve for $g \equiv \ln f$
Additive separability	$f(x_1, \dots, x_k, x_{k+1}, \dots, x_n) = g(x_1, \dots, x_k) + h(x_{k+1}, \dots, x_n)$	Solve for g & h
Multiplicative separability	$f(x_1, \dots, x_k, x_{k+1}, \dots, x_n) = g(x_1, \dots, x_k)h(x_{k+1}, \dots, x_n)$	Solve for g & h
Simple symmetry	$f(x_1, x_2, \dots) = g(x_1 \odot x_2, \dots)$, $\odot \in \{+, -, \times, /\}$	Solve for g
Compositionality	$f(x_1, \dots, x_n) = g(h(x_1, \dots, x_n))$, h simpler than f	Find h with $\nabla h \propto \nabla f$
Generalized symmetry	$f(x_1, \dots, x_k, x_{k+1}, \dots, x_n) = g[h(x_1, \dots, x_k), x_{k+1}, \dots, x_n]$	Find h satisfying $\frac{\partial h}{\partial x_i} \propto \frac{\partial f}{\partial x_i}$, $i = 1, \dots, k$
Generalized additivity	$f(x_1, x_2) = F[g(x_1) + h(x_2)]$	Solve for F , g & h
Zero-snap	\tilde{f} has numerical parameters \mathbf{p}	Replace p_i by 0
Integer snap	\tilde{f} has numerical parameters \mathbf{p}	Round p_i to integer
Rational snap	\tilde{f} has numerical parameters \mathbf{p}	Round p_i to fraction
Reoptimize	\tilde{f} has numerical parameters \mathbf{p}	Reoptimize \mathbf{p} to minimize inaccuracy

Table 4: Test equations exhibiting translational symmetry $h = x \pm y$ (T), scaling symmetry $h = x/y$ (S), product symmetry $h = xy$ (P), generalized symmetry (G), multiplicative separability (M), compositionality (C) and generalized additivity (A).

	Equation	Symmetries
1	$\delta = -5.41 + 4.9 \frac{\alpha - \beta + \gamma / \chi}{3\chi}$	TC
2	$\chi = 0.23 + 14.2 \frac{\alpha + \beta}{3\gamma}$	TS
3	$\beta = 213.80940889 (1 - e^{-0.54723748542\alpha})$	
4	$\delta = 6.87 + 11\sqrt{\alpha\beta\gamma}$	P
5	$V = [R_1^{-1} + R_2^{-1} + R_3^{-1} + R_4^{-1}]^{-1} I_0 \cos \omega t$ (Parallel resistors)	PGSM
6	$I_0 = \frac{V_0}{\sqrt{R^2 + (\omega L - \frac{1}{\omega C})^2}}$ (RLC circuit)	MG
7	$I = \frac{V_0 \cos \omega t}{\sqrt{R^2 + (\omega L - \frac{1}{\omega C})^2}}$ (RLC circuit)	MG
8	$V_2 = (\frac{R_2}{R_1 + R_2} - \frac{R_x}{R_x + R_3}) V_1$ (Wheatstone bridge)	SGMA
9	$v = c \frac{(v_1 + v_2 + v_3)/c + v_1 v_2 v_3 / c^3}{1 + (v_1 v_2 + v_1 v_3 + v_2 v_3)/c^2}$ (Velocity addition)	AG
10	$v = c \frac{(v_1 + v_2 + v_3 + v_4)/c + (v_2 v_3 v_4 + v_1 v_3 v_4 + v_1 v_2 v_4 + v_1 v_2 v_3)/c^3}{1 + (v_1 v_2 + v_1 v_3 + v_1 v_4 + v_2 v_3 + v_2 v_4 + v_3 v_4)/c^2 + v_1 v_2 v_3 v_4 / c^4}$ (Velocity addition)	GA
11	$z = (x^4 + y^4)^{1/4}$ (L_4 -norm)	AC
12	$w = xyz - z\sqrt{1-x^2}\sqrt{1-y^2} - y\sqrt{1-x^2}\sqrt{1-z^2} - x\sqrt{1-y^2}\sqrt{1-z^2}$	GA
13	$z = \frac{xy + \sqrt{1-x^2-y^2+x^2y^2}}{y\sqrt{1-x^2} - x\sqrt{1-y^2}}$	A
14	$z = y\sqrt{1-x^2} + x\sqrt{1-y^2}$	A
15	$z = xy - \sqrt{1-x^2}\sqrt{1-y^2}$	A
16	$r = \frac{a}{\cot(\alpha/2) + \cot(\beta/2)}$ (Incircle)	GMAC

Network Training



The neural network is a fully-connected, feed-forward neural network with 4 hidden layers of 128, 128, 64 and 64 neurons, respectively, all with **tanh** activation function.

Extension: Symbolic Regression for Probability Distribution

Leveraging “normalizing flows” to symbolic regress “probability distributions”

An important but more difficult symbolic regression problem is when the unknown function $f(x)$ is a probability distribution from which we have random samples \mathbf{x}_i rather than direct evaluations $f(\mathbf{x}_i)$.

We tackle this by adding preceding the regression by a step that estimates $f(\mathbf{x})$.

For this step, we use the popular “normalizing flow” technique, training an invertible neural network mapping $\mathbf{x} \rightarrow \mathbf{x}' = g(\mathbf{x})$ such that \mathbf{x}' has a multivariate normal distribution $n(\mathbf{x}')$.

We then obtain our estimator $f_{NN}(\mathbf{x}) = n[g(\mathbf{x})] |J|$ where J is the Jacobian of g .

Extension: Symbolic Regression for Probability Distribution

Table 5: Probability distributions and number of samples N required to discover them

Distribution Name	Probability distribution	N
Laplace distribution	$\frac{1}{2}e^{- x }$	10^2
Beta distribution ($\alpha = 0.5, \beta = 0.5$)	$\frac{1}{\pi} \frac{1}{\sqrt{x(1-x)}}$	10^4
Beta distribution ($\alpha = 5, \beta = 2$)	$30x^4(1-x)$	10^4
Harmonic oscillator ($n = 2, \frac{m\omega}{\hbar} = 1$)	$\frac{2}{\sqrt{\pi}}x^2e^{-x^2}$	10^5
Sinc diffraction pattern	$\frac{1}{\pi} \left(\frac{\sin x}{x}\right)^2$	10^4
2D normal distribution (correlated)	$\frac{1}{\sqrt{3}\pi}e^{-\frac{2}{3}(x^2-xy+y^2)}$	10^3
2D harmonic oscillator ($n = 2, m = 1, \frac{m\omega}{\hbar} = 1$)	$\frac{2}{\pi}x^2e^{-x^2-y^2}$	10^5
Hydrogen orbital ($n = 1, l = 0, m = 0$)	$\frac{1}{\pi}e^{-2r}$	10^3
Hydrogen orbital ($n = 2, l = 1, m = 0$)	$\frac{1}{16}r^2e^{-r}\cos^2\theta$	-
Hydrogen orbital ($n = 3, l = 1, m = 0$)	$\frac{1}{729}r^2\left(4 - \frac{2r}{3}\right)^2e^{-\frac{2r}{3}}\cos^2\theta$	-

Some related papers:

- 1) Distilling Free-Form Natural Laws from Experimental Data, by Michael Schmidt and Hod Lipson, Science, vol. 324, pp. 81-85, 2009.
- 2) Supporting Online Material for “Distilling Free-Form Natural Laws from Experimental Data,” by Michael Schmidt and Hod Lipson, Science, vol. 324, pp. 81-85, April 3, 2009.
- 3) Function Finding and the Creation of Numerical Constants in Gene Expression Programming, by Candida Ferreira, AICS 2003.
- 4) Coevolution of Fitness Predictors, by Michael D. Schmidt and Hod Lipson, IEEE Transactions on Evolutionary Computation, vol. 12, no. 6, pp. 736-749, December 2008.
- 5) Learning Equations for Extrapolation and Control, by Subham S. Sahoo, Christoph H. Lampert, and Georg Martius, 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.
- 6) Deep Symbolic Regression: Discovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients, by Brenden K. Petersen, Mikel Landajuela Larra, T. Nathan Mundhenk, Claudio Santiago, Sookyoung Kim and Joanne T. Kim, at ICLR 2021.
- 7) AI Feynman: A physics-inspired method for symbolic regression, by Silviu-Marian Udrescu and Max Tegmark, in Science Advances, 6 (16), eaay: 2631, April 15, 2020.
- 8) AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity, by Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark, in 34th Conference on Neural Information Processing Systems (NeurIPS), Vancouver, Canada, 2020.

Questions are welcome!