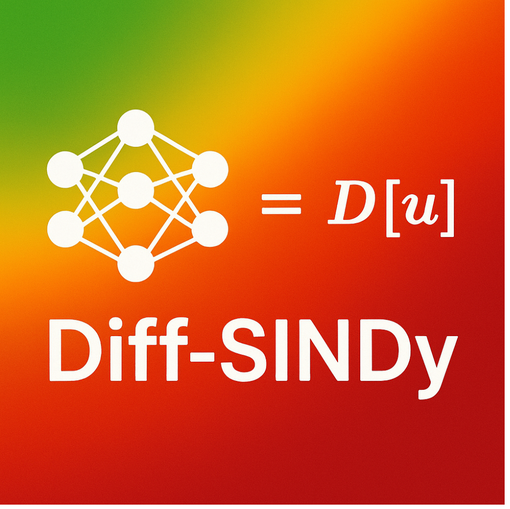


UNIVERSITY OF MICHIGAN

Diff-SINDy: A Differentiable framework for Discovering Partial Differential Equations from Data

Eddie Aljamal, Thomas M. Baer, and Devon Loomis

Introduction

- Discovering the governing equations of complex nonlinear partial differential equations (PDEs) from data remains a fundamental challenge across scientific domains.

Introduction

- Discovering the governing equations of complex nonlinear partial differential equations (PDEs) from data remains a fundamental challenge across scientific domains.
- Traditional approaches such as SINDy [1] and PINN-SR [2] rely on fixed libraries of candidate functions, limiting flexibility.

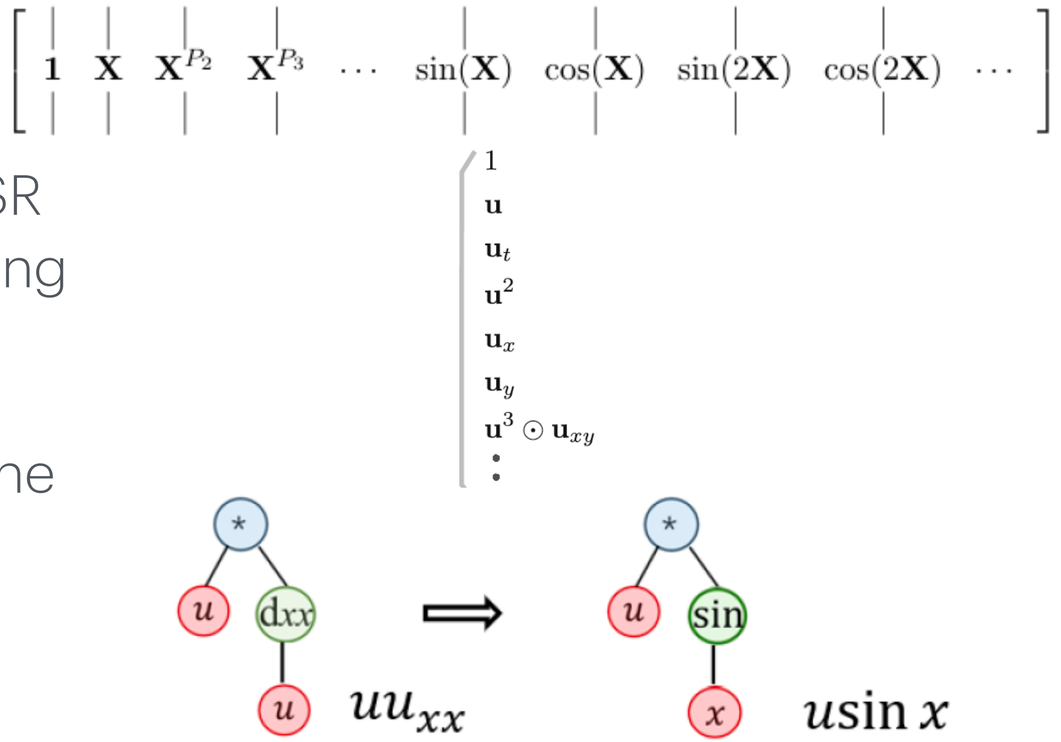
$$\begin{bmatrix} | & | & | & | & | & | & | & | & | \\ \mathbf{1} & \mathbf{X} & \mathbf{X}^{P_2} & \mathbf{X}^{P_3} & \cdots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \sin(2\mathbf{X}) & \cos(2\mathbf{X}) & \cdots \\ | & | & | & | & | & | & | & | & | \end{bmatrix}$$

$$\begin{matrix} 1 \\ \mathbf{u} \\ \mathbf{u}_t \\ \mathbf{u}^2 \\ \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}^3 \odot \mathbf{u}_{xy} \\ \vdots \end{matrix}$$

[1] Brunton et al. (2016), [2] Chen et al (2021)

Introduction

- Discovering the governing equations of complex nonlinear partial differential equations (PDEs) from data remains a fundamental challenge across scientific domains.
- Traditional approaches such as SINDy [1] and PINN-SR [2] rely on fixed libraries of candidate functions, limiting flexibility.
- Evo-SINDy [3] relaxes this constraint by introducing the derivative as a unary operator but remains non-differentiable in training and depends on discrete derivative estimates that are sensitive to noise



[1] Brunton et al. (2016), [2] Chen et al (2021), [3] Jiang et al. (2025)

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.
- Why is differentiability so important?

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅

Introduction

Continued…

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.
- Why is differentiability so important?
    1. Allows for gradient based learning.
    2. Continuous optimization that can be integrated to other ML frameworks.

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.
- Why is differentiability so important?
    1. Allows for gradient based learning.
    2. Continuous optimization that can be integrated to other ML frameworks.

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅
Differentiability	❌	✅	❌	✅

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.
- Why is differentiability so important?
    1. Allows for gradient based learning.
    2. Continuous optimization that can be integrated to other ML frameworks.
- Automatic differentiation (AD) allows for derivatives to be calculated without a mesh, interpolation in a noise-resilient manner.

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅
Differentiability	❌	✅	❌	✅

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.
- Why is differentiability so important?
    1. Allows for gradient based learning.
    2. Continuous optimization that can be integrated to other ML frameworks.
- Automatic differentiation (AD) allows for derivatives to be calculated without a mesh, interpolation in a noise-resilient manner.

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅
Differentiability	❌	✅	❌	✅
AD derivatives	❌	✅	❌	✅

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.
- Why is differentiability so important?
    1. Allows for gradient based learning.
    2. Continuous optimization that can be integrated to other ML frameworks.
- Automatic differentiation (AD) allows for derivatives to be calculated without a mesh, interpolation in a noise-resilient manner.
- Sparsity allows for parsimonious PDEs which occur in physical systems.

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅
Differentiability	❌	✅	❌	✅
AD derivatives	❌	✅	❌	✅
Sparsity	✅	✅	✅	✅

Introduction

Continued...

- Libraries limit the flexibility of equation discovery and a sufficiently large library requires a lot of memory to be held at run-time.
- Why is differentiability so important?
    1. Allows for gradient based learning.
    2. Continuous optimization that can be integrated to other ML frameworks.
- Automatic differentiation (AD) allows for derivatives to be calculated without a mesh, interpolation in a noise-resilient manner.
- Sparsity allows for parsimonious PDEs which occur in physical systems.

	SINDy	PINN-SR	Evo-SINDy	**Diff-SINDy**
No Functional library	❌	❌	✅	✅
Differentiability	❌	✅	❌	✅
AD derivatives	❌	✅	❌	✅
Sparsity	✅	✅	✅	✅
No pretaining	✅	❌	❌	✅

Diff-SINDy

What is Diff-SINDy?

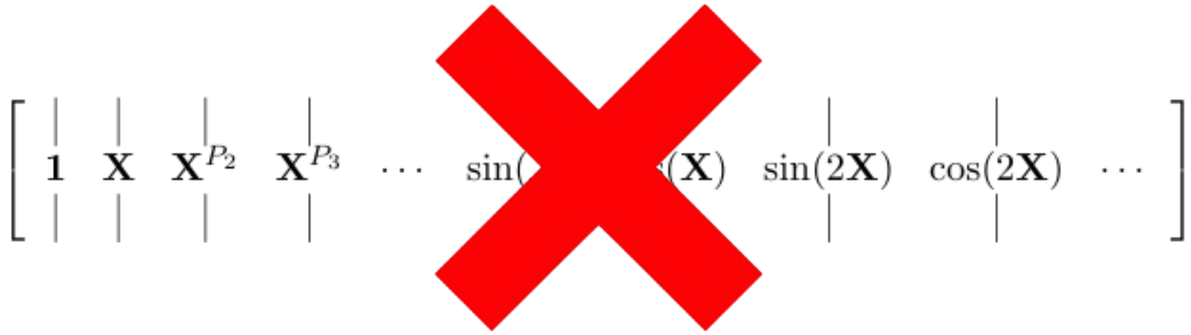
- **Diff-SINDy is a differentiable framework for data-driven PDE discovery,** separating fully-differentiable training and Symbolic Regression (SR) inference stages

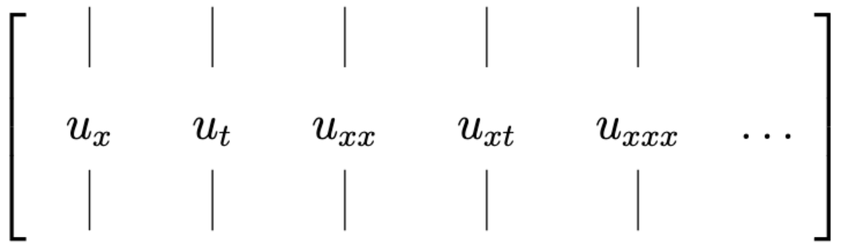
Diff-SINDy

What is Diff-SINDy?

- **Diff-SINDy is a differentiable framework for data-driven PDE discovery,** separating fully-differentiable training and Symbolic Regression (SR) inference stages

- A functional library is replaced by a derivative library, which is more flexible.



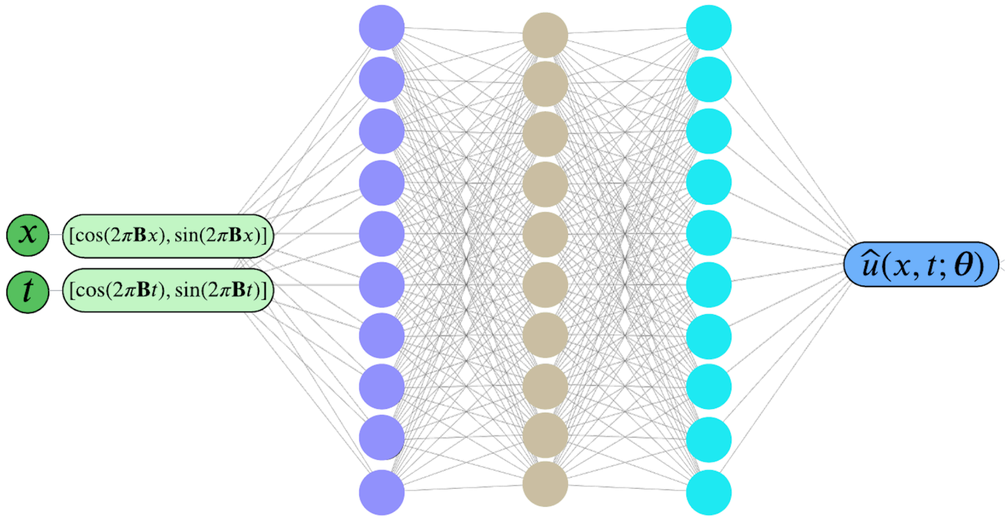


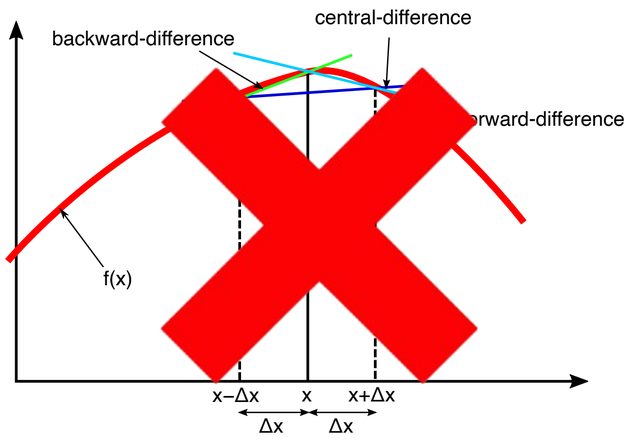
[1] Brunton et al. (2016)

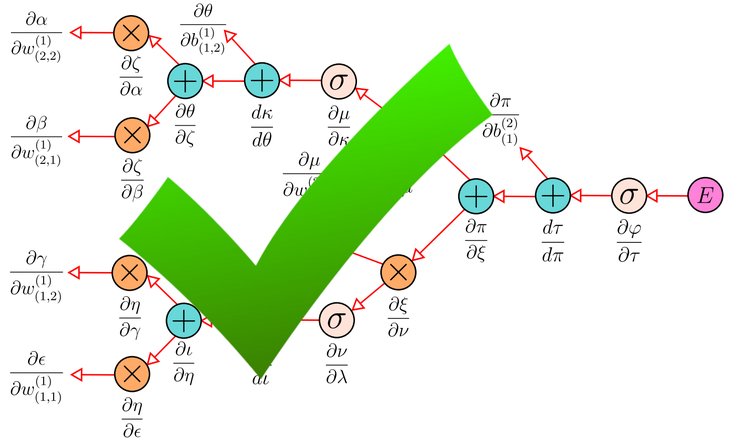
Diff-SINDy

What is Diff-SINDy?

- **Diff-SINDy is a differentiable framework for data-driven PDE discovery,** separating fully-differentiable training and Symbolic Regression (SR) inference stages
- A functional library is replaced by a derivative library, which is more flexible.
- The solution is approximated using a deep neural network (DNN) with derivatives using AD.



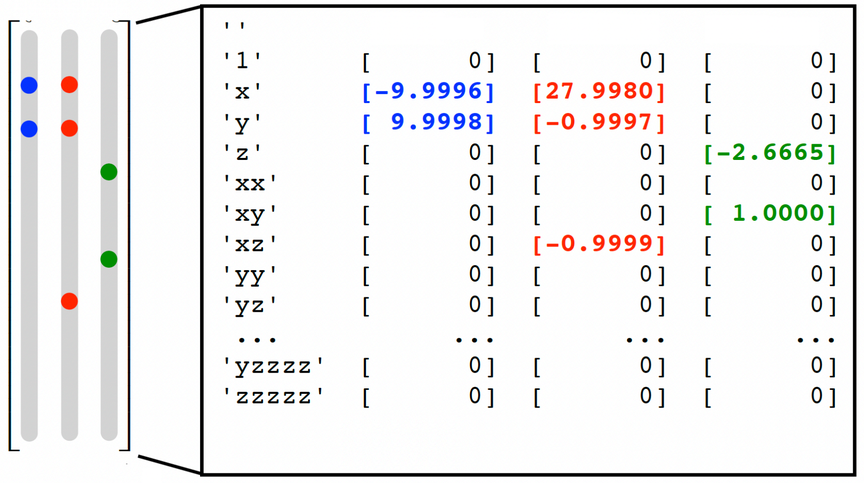




Diff-SINDy

What is Diff-SINDy?

- **Diff-SINDy is a differentiable framework for data-driven PDE discovery,** separating fully-differentiable training and Symbolic Regression (SR) inference stages
- A functional library is replaced by a derivative library, which is more flexible.
- The solution is approximated using a deep neural network (DNN) with derivatives using AD.
- Sparsity is promoted with Sequentially Thresholded Ridge Regression (SINDy-like)



[1] Brunton et al. (2016)

Problem Formulation

What we are trying to accomplish

- Though our method can be extended to general PDEs, we focus on PDEs that are linear in derivatives (no nonlinear functions of derivatives) with arbitrary coefficient function:

$$u_t = f_0(x,t,u) + f_1(x,t,u)\,u_x + f_2(x,t,u)\,u_{xx} + f_3(x,t,u)\,u_{tt} + \cdots + f_j(x,t,u)\,u_{xxx} + \ldots$$

$$u_{tt} = g_0(x,t,u) + g_1(x,t,u)\,u_x + g_2(x,t,u)\,u_t + g_3(x,t,u)\,u_{xx} + \cdots + g_j(x,t,u)\,u_{xxx} + \ldots$$

Problem Formulation

What we are trying to accomplish

- Though our method can be extended to general PDEs, we focus on PDEs that are linear in derivatives (no nonlinear functions of derivatives) with arbitrary coefficient function:

$$u_t = f_0(x,t,u) + f_1(x,t,u)\,u_x + f_2(x,t,u)\,u_{xx} + f_3(x,t,u)\,u_{tt} + \cdots + f_j(x,t,u)\,u_{xxx} + \ldots$$

$$u_{tt} = g_0(x,t,u) + g_1(x,t,u)\,u_x + g_2(x,t,u)\,u_t + g_3(x,t,u)\,u_{xx} + \cdots + g_j(x,t,u)\,u_{xxx} + \ldots$$

- The derivatives are specified in a derivative library based on minimal prior knowledge.

Problem Formulation

What we are trying to accomplish

- Though our method can be extended to general PDEs, we focus on PDEs that are linear in derivatives (no nonlinear functions of derivatives) with arbitrary coefficient function:

$$u_t = f_0(x,t,u) + f_1(x,t,u)\,u_x + f_2(x,t,u)\,u_{xx} + f_3(x,t,u)\,u_{tt} + \cdots + f_j(x,t,u)\,u_{xxx} + \ldots$$

$$u_{tt} = g_0(x,t,u) + g_1(x,t,u)\,u_x + g_2(x,t,u)\,u_t + g_3(x,t,u)\,u_{xx} + \cdots + g_j(x,t,u)\,u_{xxx} + \ldots$$

- The derivatives are specified in a derivative library based on minimal prior knowledge.

- Why minimal prior knowledge?

Problem Formulation

What we are trying to accomplish

- Though our method can be extended to general PDEs, we focus on PDEs that are linear in derivatives (no nonlinear functions of derivatives) with arbitrary coefficient function:

$$u_t = f_0(x,t,u) + f_1(x,t,u)\,u_x + f_2(x,t,u)\,u_{xx} + f_3(x,t,u)\,u_{tt} + \cdots + f_j(x,t,u)\,u_{xxx} + \ldots$$

$$u_{tt} = g_0(x,t,u) + g_1(x,t,u)\,u_x + g_2(x,t,u)\,u_t + g_3(x,t,u)\,u_{xx} + \cdots + g_j(x,t,u)\,u_{xxx} + \ldots$$

- The derivatives are specified in a derivative library based on minimal prior knowledge.

- Why minimal prior knowledge?

Most PDE contain a few derivatives with each derivative being of modest order. **This is the utility of replacing a functional library with a derivative library.**

Problem Formulation

What we are trying to accomplish

- Though our method can be extended to general PDEs, we focus on PDEs that are linear in derivatives (no nonlinear functions of derivatives) with arbitrary coefficient function:

$$u_t = f_0(x,t,u) + f_1(x,t,u)\,u_x + f_2(x,t,u)\,u_{xx} + f_3(x,t,u)\,u_{tt} + \cdots + f_j(x,t,u)\,u_{xxx} + \ldots$$

$$u_{tt} = g_0(x,t,u) + g_1(x,t,u)\,u_x + g_2(x,t,u)\,u_t + g_3(x,t,u)\,u_{xx} + \cdots + g_j(x,t,u)\,u_{xxx} + \ldots$$

- The derivatives are specified in a derivative library based on minimal prior knowledge.

- Why minimal prior knowledge?

  Most PDE contain a few derivatives with each derivative being of modest order. **This is the utility of replacing a functional library with a derivative library.**

- **Our goal is to find parsimonious symbolic expressions for the coefficients, f and g, from the data.**

Burgers	$u_t = -uu_x + 0.1u_{xx}$
KdV	$u_t = -uu_x - 0.0025u_{xxx}$
Chafee-Infante	$u_t = u_{xx} - u + u^3$
Allen-Cahn	$u_t = 0.003u_{xx} + u - u^3$
KS	$u_t = -uu_{xx} - u_{xx} - u_{xxxx}$
Convection-diffusion	$u_t = -u_x + 0.25u_{xx}$
Parametric CDE	$u_t = -(1 + 0.25 \sin x)u_x + u_{xx}$
PDE_divide	$u_t = -\frac{u_x}{x} + 0.25u_x$
Custom-sin	$u_t = -u_{xx} - 5u \sin u$
Sine-Gordon	$u_{tt} = 4u_{xx} - 1.25 \sin u$
Wave	$u_{tt} = u_{xx}$
KG	$u_{tt} = 0.5u_{xx} - 5u$

Burgers	$u_t = -uu_x + 0.1u_{xx}$
KdV	$u_t = -uu_x - 0.0025u_{xxx}$
Chafee-Infante	$u_t = u_{xx} - u + u^3$
Allen-Cahn	$u_t = 0.003u_{xx} + u - u^3$
KS	$u_t = -uu_{xx} - u_{xx} - u_{xxxx}$
Convection-diffusion	$u_t = -u_x + 0.25u_{xx}$
Parametric CDE	$u_t = -(1 + 0.25\sin x)u_x + u_{xx}$
PDE_divide	$u_t = -\frac{u_x}{x} + 0.25u_x$
Custom-sin	$u_t = -u_{xx} - 5u\sin u$
Sine-Gordon	$u_{tt} = 4u_{xx} - 1.25\sin u$
Wave	$u_{tt} = u_{xx}$
KG	$u_{tt} = 0.5u_{xx} - 5u$

Whoops!

Almost forgot there are fluid dynamicists here...

Burgers	$u_t = -uu_x + 0.1u_{xx}$
KdV	$u_t = -uu_x - 0.0025u_{xxx}$
Chafee-Infante	$u_t = u_{xx} - u + u^3$
Allen-Cahn	$u_t = 0.003u_{xx} + u - u^3$
KS	$u_t = -uu_{xx} - u_{xx} - u_{xxxx}$
Convection-diffusion	$u_t = -u_x + 0.25u_{xx}$
Parametric CDE	$u_t = -(1 + 0.25\sin x)u_x + u_{xx}$
PDE_divide	$u_t = -\frac{u_x}{x} + 0.25u_x$
Custom-sin	$u_t = -u_{xx} - 5u\sin u$
Sine-Gordon	$u_{tt} = 4u_{xx} - 1.25\sin u$
Wave	$u_{tt} = u_{xx}$
KG	$u_{tt} = 0.5u_{xx} - 5u$

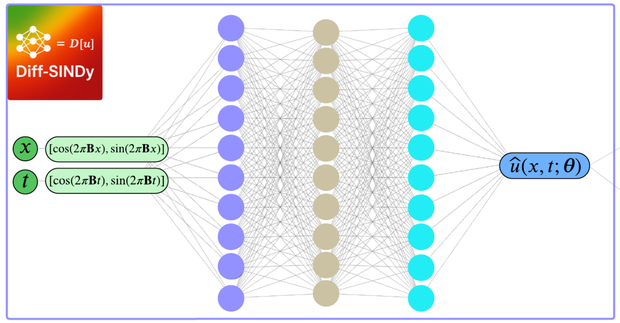
Whoops!

Almost forgot there are fluid dynamicists here...

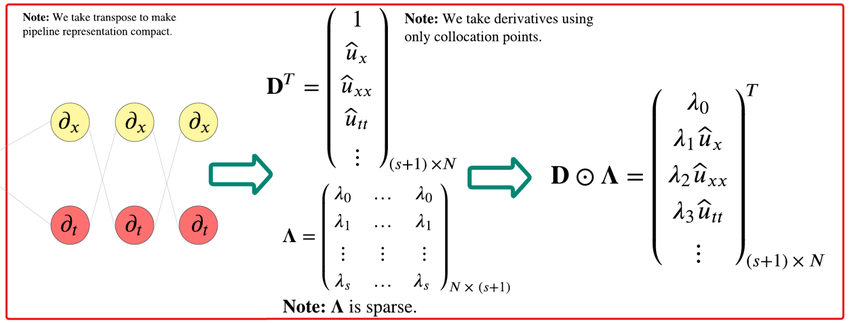
NS $u_t = -uu_x - \frac{1}{\rho}p_x + \nu u_{xx} + f(x,t)$

Methods: Diff-SINDy Training

a. DNN with parameters $\boldsymbol{\theta}$



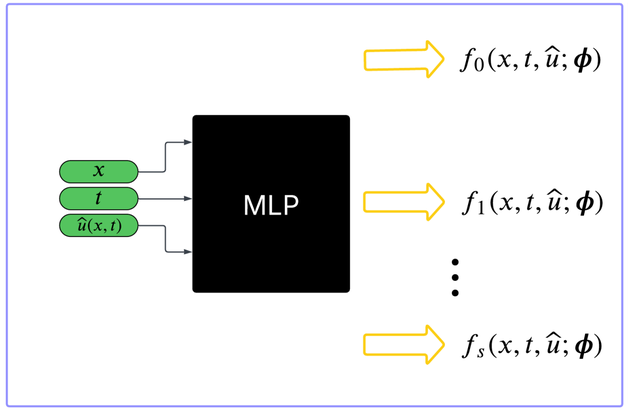
Automatic Differentiation with sparsity $\boldsymbol{\Lambda}$



$$\mathbf{D}^T = \begin{pmatrix} 1 \\ \hat{u}_x \\ \hat{u}_{xx} \\ \hat{u}_{tt} \\ \vdots \end{pmatrix}_{(s+1)\times N} \qquad \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_0 & \dots & \lambda_0 \\ \lambda_1 & \dots & \lambda_1 \\ \vdots & \vdots & \vdots \\ \lambda_s & \dots & \lambda_s \end{pmatrix}_{N\times(s+1)}$$

$$\mathbf{D} \odot \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_0 \\ \lambda_1 \hat{u}_x \\ \lambda_2 \hat{u}_{xx} \\ \lambda_3 \hat{u}_{tt} \\ \vdots \end{pmatrix}^T_{(s+1)\times N}$$

b. MLP with parameters $\boldsymbol{\phi}$



PDE construction

$$\mathbf{F} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_s \end{pmatrix}^T_{(s+1)\times N}$$

$$\frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - (\mathbf{D} \odot \boldsymbol{\Lambda} \odot \mathbf{F})\, \mathbf{1}_{(s+1)\times 1} \right\|_2^2 \to 0$$

OR

$$\frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - \sum_{j=1}^{s+1} (\mathbf{D} \odot \boldsymbol{\Lambda})_{:,j} \odot \mathbf{F}_{:,j} \right\|_2^2 \to 0$$

c. Losses and Training

$$\mathcal{L}_{\text{IC}}(\boldsymbol{\theta}; \overbrace{u_{\text{IC}}}^{\text{measured}}) = \frac{1}{N_{\text{IC}}} \| \overbrace{\hat{u}_{\text{IC}}}^{\text{predicted}} - u_{\text{IC}} \|_2^2 \qquad \mathcal{L}_{\text{Data}}(\boldsymbol{\theta}; u_{\text{Data}}) = \frac{1}{N_{\text{Data}}} \| \hat{u}_{\text{Data}} - u_{\text{Data}} \|_2^2 \qquad \mathcal{L}_{\text{BC}}(\boldsymbol{\theta}; u_{\text{BC}}) = \frac{1}{N_{\text{BC}}} \| \hat{u}_{\text{BC}} - u_{\text{BC}} \|_2^2$$

$$\mathcal{L}_{\text{PDE}}(\boldsymbol{\phi}; x_{\text{Col.}}, t_{\text{Col.}}, u_{\text{Col.}}, \boldsymbol{\Lambda}) = \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - (\mathbf{D} \odot \boldsymbol{\Lambda} \odot \mathbf{F})\, \mathbf{1}_{(s+1)\times 1} \right\|_2^2$$

Stage 1: Train solution with frozen sparsity and frozen coefficients model

$$\mathcal{L}(\overbrace{\boldsymbol{\theta}}^{\text{unfrozen}}; x, t, u, \overbrace{\boldsymbol{\phi}, \boldsymbol{\Lambda}}^{\text{frozen}}) = \lambda_{\text{IC}} \mathcal{L}_{\text{IC}} + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}} + \lambda_{\text{Data}} \mathcal{L}_{\text{Data}} + \lambda_{\text{PDE}}(\text{epoch}) \mathcal{L}_{\text{PDE}}$$

Stage 2: Train coefficient model and sparsity with **STRidge** using collocation points

$$\mathcal{L}(\overbrace{\boldsymbol{\phi}, \boldsymbol{\Lambda}}^{\text{unfrozen}}; x_{\text{Col.}}, t_{\text{Col.}}, u_{\text{Col.}}, \overbrace{\boldsymbol{\theta}}^{\text{frozen}}) = \mathcal{L}_{\text{PDE}} + \beta \|\boldsymbol{\Lambda}\|_0$$

Training

Find a solution

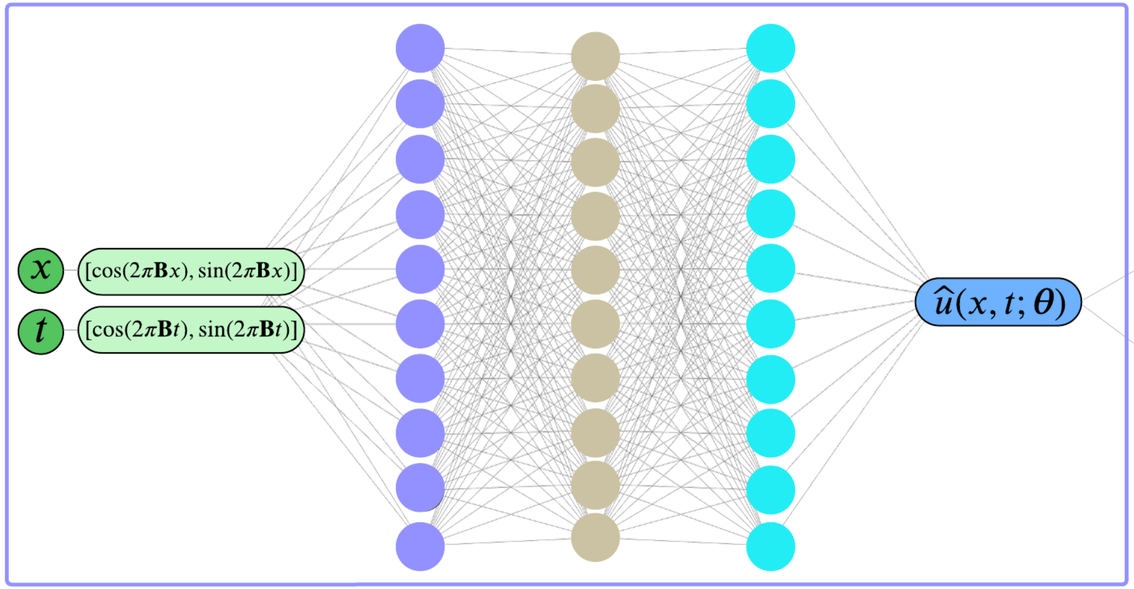
- Train a DNN to approximate the solution using an initial condition (IC), boundary condition (BC), and interior data (Data) losses.

$$\mathcal{L}_{\mathrm{IC}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{IC}}} \|\hat{u}_{\mathrm{IC}} - u_{\mathrm{IC}}\|_2^2,$$

$$\mathcal{L}_{\mathrm{BC}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{BC}}} \|\hat{u}_{\mathrm{BC}} - u_{\mathrm{BC}}\|_2^2,$$

$$\mathcal{L}_{\mathrm{Data}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{Data}}} \|\hat{u}_{\mathrm{Data}} - u_{\mathrm{Data}}\|_2^2,$$

a. DNN with parameters $\boldsymbol{\theta}$



Training

Find a solution

- Train a DNN to approximate the solution using an initial condition (IC), boundary condition (BC), and interior data (Data) losses.

$$\mathcal{L}_{\mathrm{IC}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{IC}}} \|\hat{u}_{\mathrm{IC}} - u_{\mathrm{IC}}\|_2^2,$$

$$\mathcal{L}_{\mathrm{BC}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{BC}}} \|\hat{u}_{\mathrm{BC}} - u_{\mathrm{BC}}\|_2^2,$$

$$\mathcal{L}_{\mathrm{Data}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{Data}}} \|\hat{u}_{\mathrm{Data}} - u_{\mathrm{Data}}\|_2^2,$$

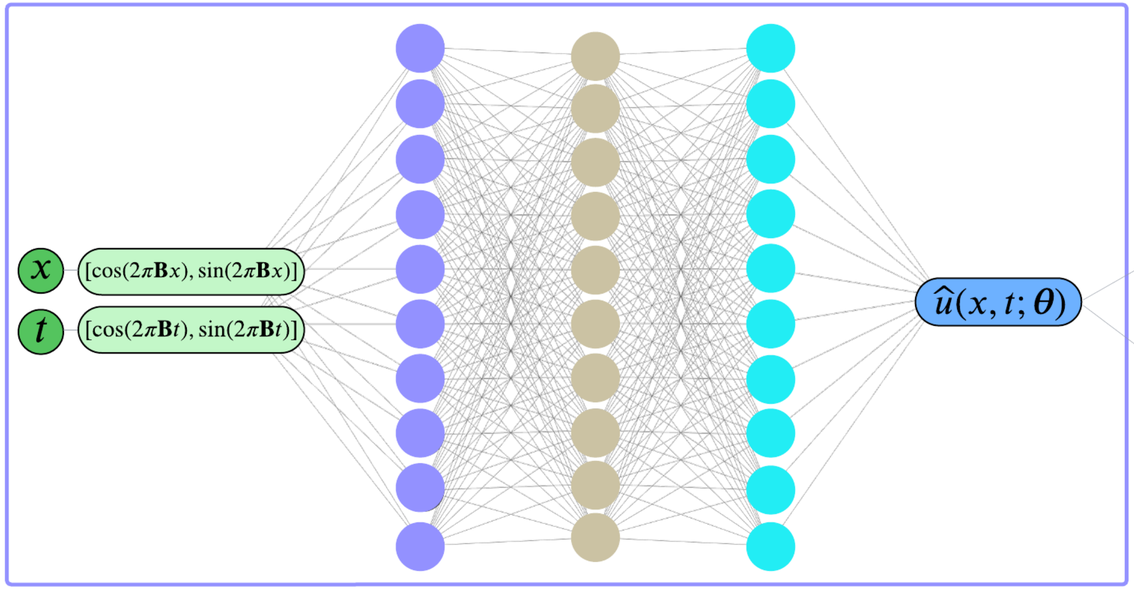
- Before beginning the two-stage training procedure, we warm the solution network, by minimizing the loss:

$$\mathcal{L}_{\mathrm{warm}} = \lambda_{\mathrm{IC}}\mathcal{L}_{\mathrm{IC}} + \lambda_{\mathrm{BC}}\mathcal{L}_{\mathrm{BC}} + \lambda_{\mathrm{Data}}\mathcal{L}_{\mathrm{Data}}$$

with gradient balancing

$$\frac{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{IC}}\|_2}{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{Data}}\|_2}, \quad \frac{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{BC}}\|_2}{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathrm{Data}}\|_2}.$$

a. DNN with parameters $\boldsymbol{\theta}$



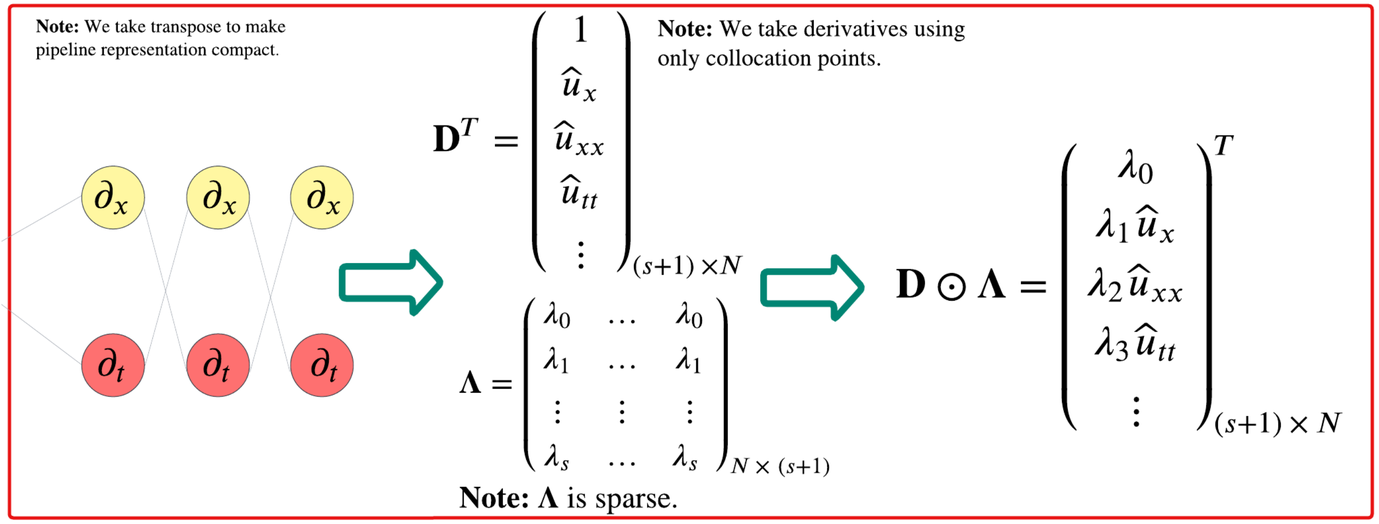
[4] Wang et al (2021)

Training

Take derivatives with sparsity

- Take derivatives using the approximate solution with AD.

Automatic Differentiation with sparsity $\mathbf{\Lambda}$

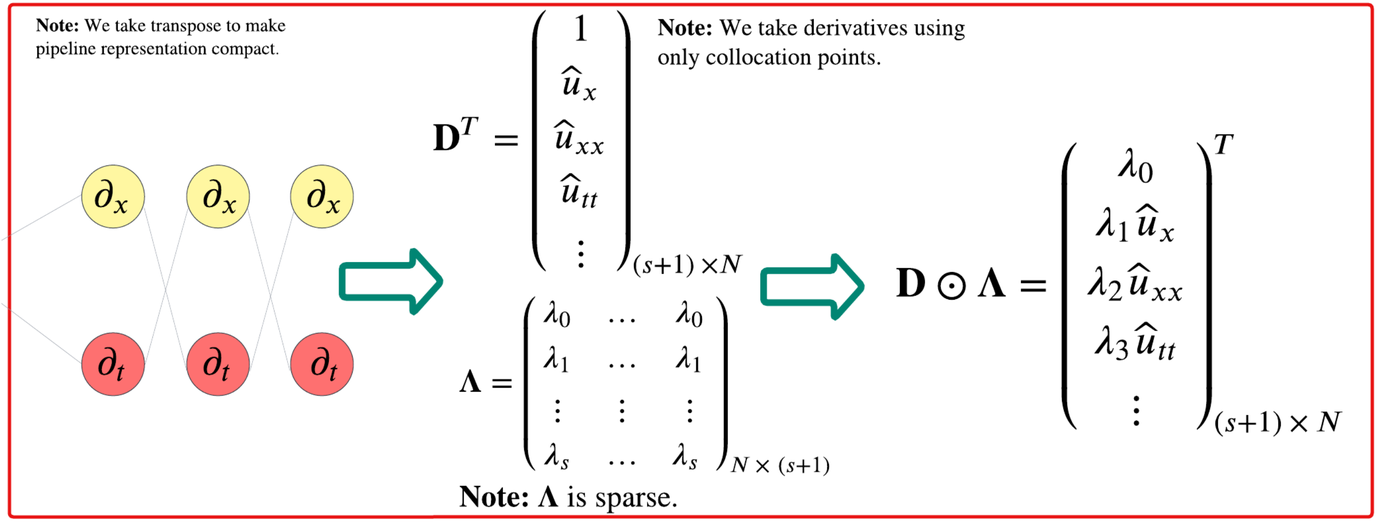


Training

Take derivatives with sparsity

- Take derivatives using the approximate solution with AD.
- Introduce a sparse matrix $\mathbf{\Lambda}$ which will have continuous parameters but is thresholded to 0s and 1s to denote non-active and active derivatives.

Automatic Differentiation with sparsity $\mathbf{\Lambda}$

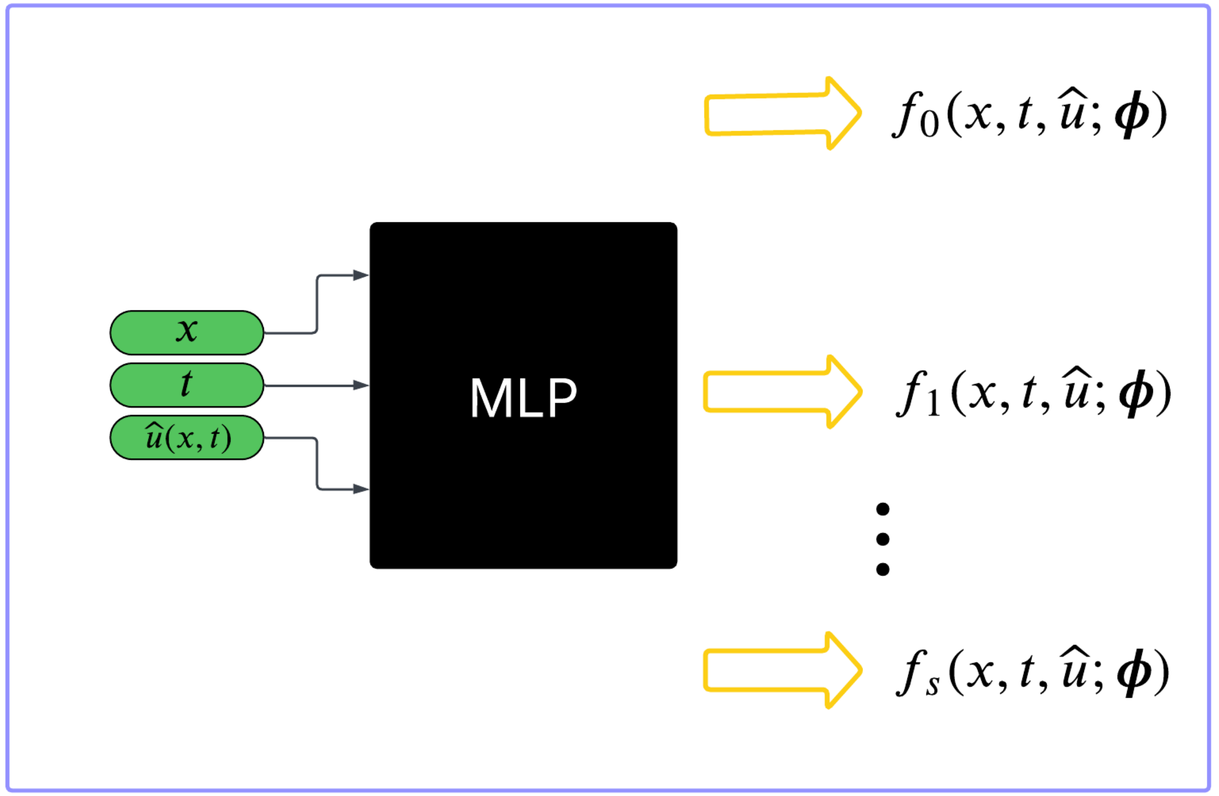


Training

Coefficient network(s)

- Compute coefficients using an MLP architecture using the spatiotemporal input and the predicted solution.

b. MLP with parameters $\boldsymbol{\phi}$

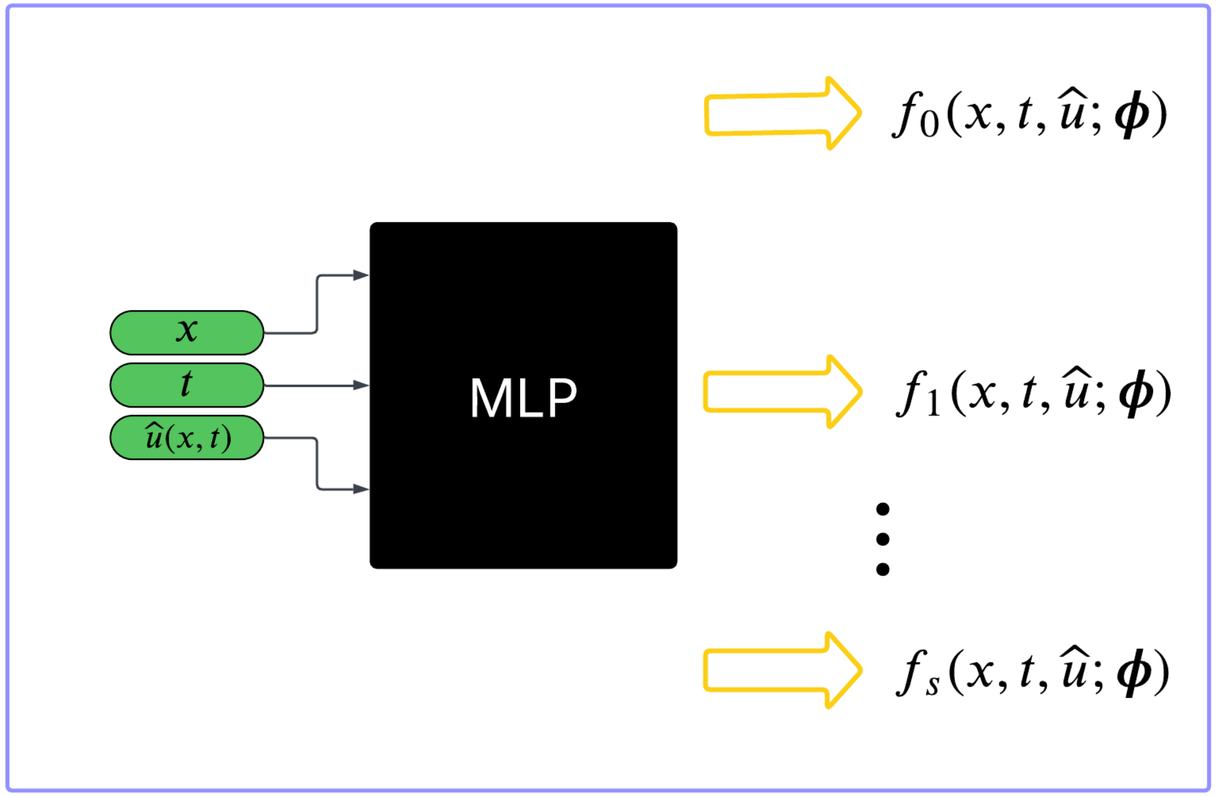


Training

Coefficient network(s)

- Compute coefficients using an MLP architecture using the spatiotemporal input and the predicted solution.
- Our code supports three types of architectures depending on the expected level of correlation between coefficients.

b. MLP with parameters $\boldsymbol{\phi}$

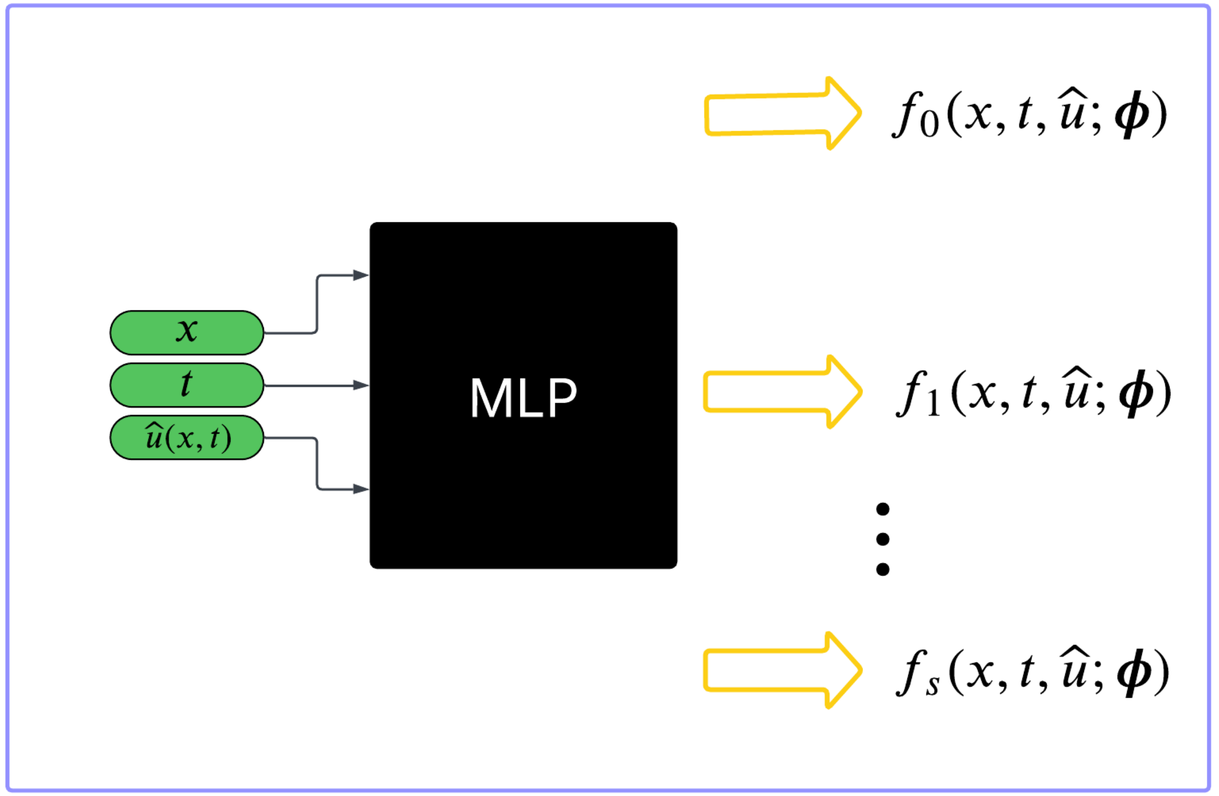


Training

Coefficient network(s)

- Compute coefficients using an MLP architecture using the spatiotemporal input and the predicted solution.
- Our code supports three types of architectures depending on the expected level of correlation between coefficients.
    1. A single MLP with many outputs.

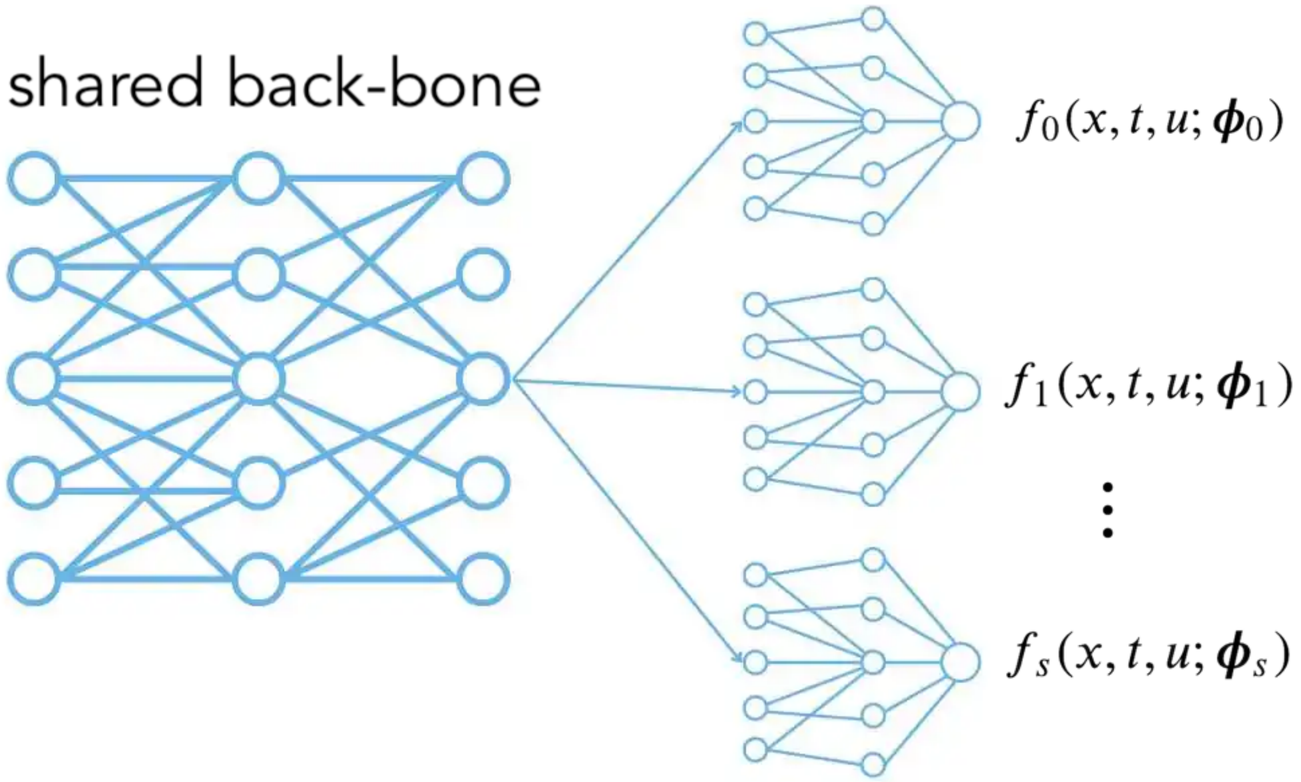
b. MLP with parameters $\boldsymbol{\phi}$



Training

Coefficient network(s)

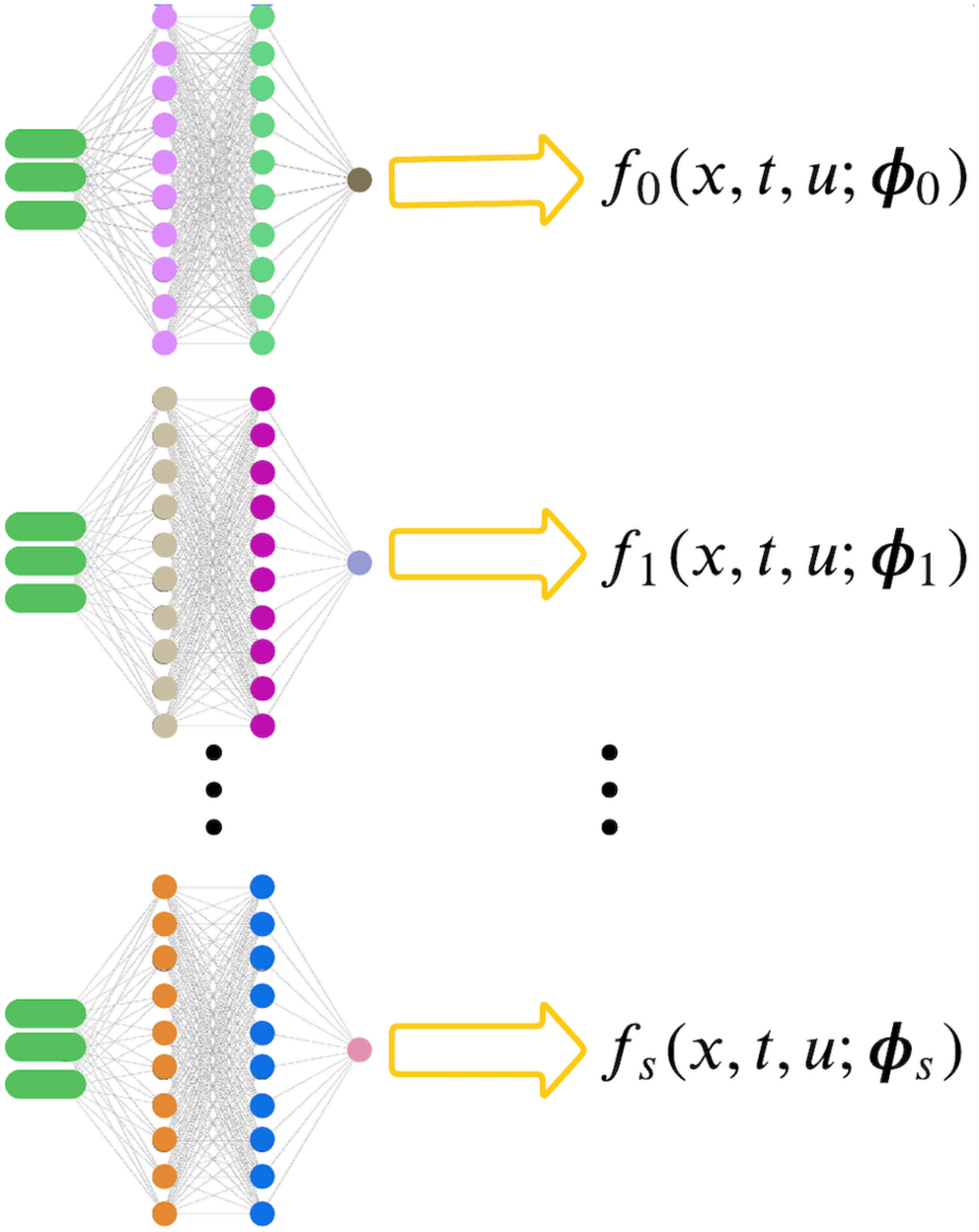
- Compute coefficients using an MLP architecture using the spatiotemporal input and the predicted solution.
- Our code supports three types of architectures depending on the expected level of correlation between coefficients.
    1. A single MLP with many outputs.
    2. Multi-task MLP with shared backbone and a separate head for each coefficient.



Training

Coefficient network(s)

- Compute coefficients using an MLP architecture using the spatiotemporal input and the predicted solution.
- Our code supports three types of architectures depending on the expected level of correlation between coefficients.
    1. A single MLP with many outputs.
    2. Multi-task MLP with shared backbone and a separate head for each coefficient.
    3. A separate NN for each coefficient.



Training

PDE construction

- PDE loss is constructed from the coefficients, derivatives and sparsity parameters.

PDE construction

$$\Rightarrow \mathbf{F} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_s \end{pmatrix}^T_{(s+1)\times N}$$

$$\Rightarrow \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - (\mathbf{D} \odot \boldsymbol{\Lambda} \odot \mathbf{F})\, \mathbf{1}_{(s+1)\times 1} \right\|_2^2 \to 0$$

OR

$$\Rightarrow \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - \sum_{j=1}^{s+1} (\mathbf{D} \odot \boldsymbol{\Lambda})_{:,j} \odot \mathbf{F}_{:,j} \right\|_2^2 \to 0$$

Training

PDE construction

- PDE loss is constructed from the coefficients, derivatives and sparsity parameters.
- Train in two stages:
    1. Stage 1: train solution network with sparsity and coefficient network(s) frozen. We introduce the PDE gradually i.e. λ_{PDE} gradually increases to 1.

PDE construction

$$\mathbf{F} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_s \end{pmatrix}^T_{(s+1)\times N} \Rightarrow \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - (\mathbf{D} \odot \boldsymbol{\Lambda} \odot \mathbf{F})\, \mathbf{1}_{(s+1)\times 1} \right\|_2^2 \to 0$$

OR

$$\Rightarrow \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - \sum_{j=1}^{s+1} (\mathbf{D} \odot \boldsymbol{\Lambda})_{:,j} \odot \mathbf{F}_{:,j} \right\|_2^2 \to 0$$

$$\mathcal{L}(\overbrace{\boldsymbol{\theta}}^{\text{unfrozen}}; x, t, u, \overbrace{\boldsymbol{\phi}, \boldsymbol{\Lambda}}^{\text{frozen}}) = \lambda_{\text{IC}} \mathcal{L}_{\text{IC}} + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}} + \lambda_{\text{Data}} \mathcal{L}_{\text{Data}} + \lambda_{\text{PDE}}(\text{epoch}) \mathcal{L}_{\text{PDE}}$$

Training

PDE construction

- PDE loss is constructed from the coefficients, derivatives and sparsity parameters.

- Train in two stages:

  1. Stage 1: train solution network with sparsity and coefficient network(s) frozen. We introduce the PDE gradually i.e. λ_{PDE} gradually increases to 1.

  2. Stage 2: The coefficient network with sparsity (STRidge) is trained with solution network frozen.

PDE construction

$$\mathbf{F} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_s \end{pmatrix}^T_{(s+1)\times N} \Rightarrow \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - (\mathbf{D} \odot \boldsymbol{\Lambda} \odot \mathbf{F})\, \mathbf{1}_{(s+1)\times 1} \right\|_2^2 \to 0$$

OR

$$\Rightarrow \frac{1}{N_{\text{Col.}}} \left\| \hat{u}_{t,\,\text{Col.}} - \sum_{j=1}^{s+1} (\mathbf{D} \odot \boldsymbol{\Lambda})_{:,j} \odot \mathbf{F}_{:,j} \right\|_2^2 \to 0$$

$$\mathcal{L}(\overbrace{\boldsymbol{\theta}}^{\text{unfrozen}}; x, t, u, \overbrace{\boldsymbol{\phi}, \boldsymbol{\Lambda}}^{\text{frozen}}) = \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{BC}}\mathcal{L}_{\text{BC}} + \lambda_{\text{Data}}\mathcal{L}_{\text{Data}} + \lambda_{\text{PDE}}(\text{epoch})\mathcal{L}_{\text{PDE}}$$

$$\mathcal{L}(\overbrace{\boldsymbol{\phi}, \boldsymbol{\Lambda}}^{\text{unfrozen}}; x_{\text{Col.}}, t_{\text{Col.}}, u_{\text{Col.}}, \overbrace{\boldsymbol{\theta}}^{\text{frozen}}) = \mathcal{L}_{\text{PDE}} + \beta\|\boldsymbol{\Lambda}\|_0$$

STRidge

Inference

Symbolic Regression with PySR

- After training with sparsity, we will have some s’ coefficients of the active derivatives.

[5] Cranmer et al (2023)

Inference

Symbolic Regression with PySR

- After training with sparsity, we will have some s' coefficients of the active derivatives.

- Coefficients are passed to the Symbolic Regression (SR) framework PySR. Inputs are spatio-temporal coordinates and the MLP approximated solution.

$$(x, t, \widehat{u}) \mapsto f_i, \quad i \in \{0, \ldots, s'\}.$$

[5] Cranmer et al (2023)

Inference

Symbolic Regression with PySR

- After training with sparsity, we will have some s' coefficients of the active derivatives.
- Coefficients are passed to the Symbolic Regression (SR) framework PySR. Inputs are spatio-temporal coordinates and the MLP approximated solution.
- PySR will minimize the MSE of the coefficient data given an expression using an evolutionary algorithm.

$$(x, t, \widehat{u}) \mapsto f_i, \quad i \in \{0, \ldots, s'\}.$$

[5] Cranmer et al (2023)

Inference

Symbolic Regression with PySR

- After training with sparsity, we will have some s' coefficients of the active derivatives.
- Coefficients are passed to the Symbolic Regression (SR) framework PySR. Inputs are spatio-temporal coordinates and the MLP approximated solution.

$$(x, t, \widehat{u}) \mapsto f_i, \quad i \in \{0, \ldots, s'\}.$$

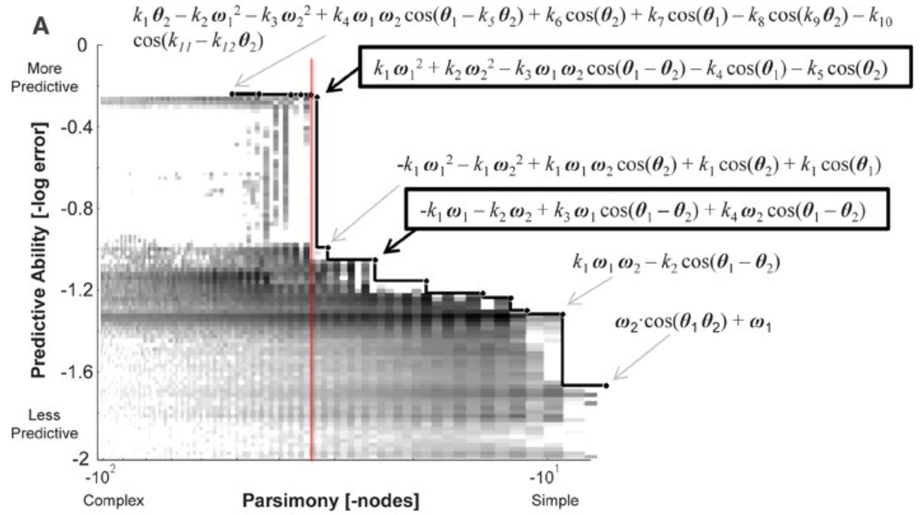
- PySR will minimize the MSE of the coefficient data given an expression using an evolutionary algorithm.
- To guide the expression selection, we use two measures. MSE loss quantifies fit quality but can favor overly complex expressions. Second, a **score** balances fit and complexity by estimating the trade-off between loss reduction.

$$\text{score}_i = -\frac{\log(\text{loss}_i - \text{loss}_{i-1})}{\text{complexity}_i - \text{complexity}_{i-1}}.$$

[5] Cranmer et al (2023)

Inference

Symbolic Regression with PySR



- After training with sparsity, we will have some s' coefficients of the active derivatives.

- Coefficients are passed to the Symbolic Regression (SR) framework PySR. Inputs are spatio-temporal coordinates and the MLP approximated solution.

$$(x, t, \widehat{u}) \mapsto f_i, \quad i \in \{0, \ldots, s'\}.$$

- PySR will minimize the MSE of the coefficient data given an expression using an evolutionary algorithm.

- To guide the expression selection, we use two measures. MSE loss quantifies fit quality but can favor overly complex expressions. Second, a **score** balances fit and complexity by estimating the trade-off between loss reduction.

$$\text{score}_i = -\frac{\log(\text{loss}_i - \text{loss}_{i-1})}{\text{complexity}_i - \text{complexity}_{i-1}}.$$

[5] Cranmer et al (2023), [6] Lipson and Schmidt

Experiments

Data

- Homogenous heat equation

$$u_t = 0.005 u_{xx}$$

IBC

$$u(x,0) = 50,$$
$$u(-1,t) = 70, \quad u(1,t) = 90$$

- Wave equation

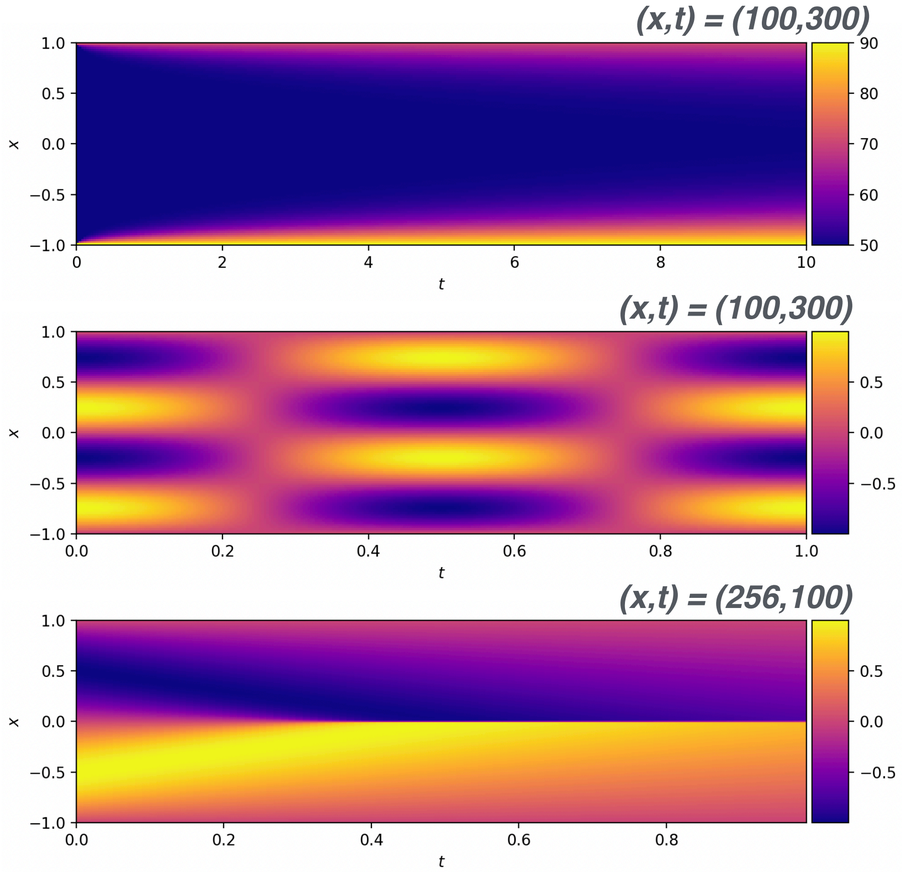
$$u_{tt} = 1 u_{xx}$$

$$u(x,0) = \sin(2\pi x),$$
$$u(-1,t) = u(1,t) = 0$$

- Burgers equation

$$u_t = -1 u u_x + \frac{0.01}{\pi} u_{xx}$$

$$u(x,0) = -\sin(\pi x),$$
$$u(-1,t) = u(1,t) = 0$$



Experiments

Training Details

- Use 10,000 collocation points for all experiments.
- Only derivatives that are in the PDE are supplied. Sparsity is in preliminary stages but results are almost identical.
- Warm up 200 epochs. 1000 epochs in total, 500 to gradually increase λ_{PDE} to 1 during Stage 1, 500 with it set to 1.
- Training took 30 minutes for each equation on a single T4 GPU.

Inference with PySR.

- Binary operators: [*, +, -, /] with the first three having complexity of 1 and the second complexity of 2. Constants have the lowest complexity of 1.
- Unary operators: [square, cube, sin, cos, exp], with the four having complexity 2, and the last complexity of 4.
- Early stopping for SR for MSE < 1e-10 *and* complexity < 10.

Experiments

Results: Overview

	Burgers' equation	Heat equation	Wave equation
Exact	$u_t = -1uu_x + \frac{0.01}{\pi}u_{xx}$	$u_t = 0.005\,u_{xx}$	$u_{tt} = 1\,u_{xx}$
Diff-SINDy	$u_t = -0.995uu_x + 0.00298u_{xx}$	$u_t = 0.00494\,u_{xx}$	$u_{tt} = 0.997\,u_{xx}$

PDE	PINN-SR	Evo-SINDy	**Diff-SINDy**
Homogeneous heat eq.	—	—	1%
Wave eq.	—	0.01%	0.3%
Burgers' eq.	0.9%/1%	0%/2.3%	0.5%/6%

Experiments

Results: Heat equation $u_t = 0.005 u_{xx}$

Table 3: PySR results for the coefficient of u_{xx} in the homogeneous heat equation (true coefficient 0.005).

Complexity	Expression	MSE $[10^{-6}]$	Score
1	0.00494	3.871	0.000
5	$-0.0006\,x + 0.0047$	3.761	0.007
6	$0.0024\,x^2 + 0.0040$	3.392	0.103
7	$-0.0014\cos(u) + 0.006$	2.889	0.160
9	$-0.001\cos(0.38\,u) + 0.005$	2.238	0.128

Experiments

Results: Wave equation $u_{tt} = 1u_{xx}$

Table 4: PySR results for the coefficient of u_{xx} in the wave equation (true coefficient 1).

Complexity	Expression	MSE $[10^{-6}]$	Score
1	0.997	1.652	0.000
5	$\cos(0.116\,x)$	1.563	0.238
6	$-0.006\,x^2 + 0.999$	1.356	0.141
8	$-0.006(x + 0.006)^2 + 0.999$	1.355	0.001
10	$-0.006\,x^2\cos(u) + 0.999$	1.165	0.008

Experiments

Results: Burgers' equation $u_t = -1uu_x + \frac{0.01}{\pi}u_{xx}$

Table 5: PySR results for the coefficients of u_x and u_{xx} in Burgers' equation (true coefficients $-u$ and $0.01/\pi \approx 0.00318$, respectively).

Complexity	Expression	MSE	Score
u_x coefficient			
1	x	0.2787	0.000
3	$-0.995\,u$	3.451×10^{-4}	3.346
5	$-0.982\,u + 0.003$	3.392×10^{-4}	0.0099
7	$-0.958(-0.028\,x + u)$	1.835×10^{-4}	0.510
10	$-0.966\,u + 0.032(x + 0.138)^3$	1.624×10^{-4}	0.085
u_{xx} coefficient			
1	0.00298	2.148×10^{-7}	0.000
5	$0.0009x + 0.003$	1.962×10^{-7}	0.242
7	$0.004\cos(x - 0.303)$	1.334×10^{-7}	0.191
8	$0.0005(x - 0.668)^3 + 0.003$	9.729×10^{-8}	0.316
10	$(0.006\,u + 0.001)x^3 + 0.003$	8.011×10^{-8}	0.097

Noise robustness

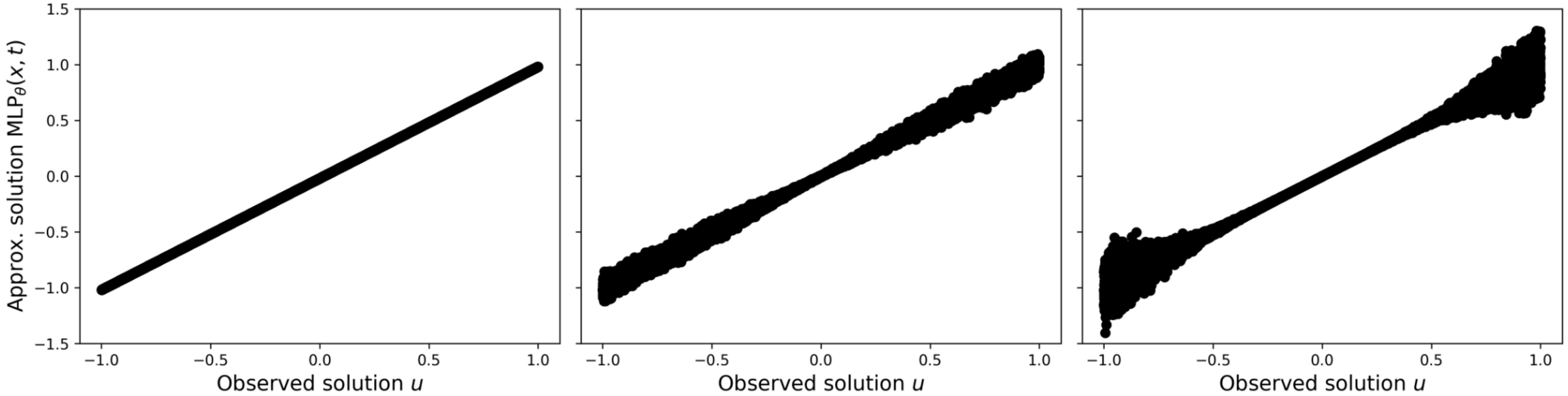


Figure 2: Predicted solutions from three differently initialized (with different architectures and settings for training) solution MLPs for Burgers' equation. Despite differences in solution accuracy, PySR recovered the same symbolic expressions (Table 4) with minor variations in MSE and score.

Conclusion

- **Diff-SINDy** is a differentiable framework for PDE discovery from Data.
  - Replaces a functional library with a derivative library.
  - Computes derivatives with automatic differentiation.
  - Promotes parsimonious PDEs that explain the data through sparsity with soft-thresholding.
  - Computationally inexpensive and requires no retraining.
- We tested Diff-SINDy on the homogeneous heat, wave, and Burgers' equations and were able to rediscover these equations with impressive accuracy.

Future Work

- We are still in active development.
- Improve stability with soft-thresholding for sparsity.
- Extend to higher dimensional PDEs and more general forms of PDEs.
- Automate the selection of constants which do not have a score measure.
- Test if pertaining on PDEs improves generalization, convergence, and efficiency.
- Study how performance scales with the size of the derivative library.
- Test for robustness to noise in coordinates and observed solutions.
- **Inject conservation laws and other constraints into proposed symbolic expressions.**

Data and Code

All data and code is available on https://github.com/ealjamal/Diff-SINDy

Thank you!

Questions?

References

- **Brunton et al. (2016):** Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *PNAS*.
- **Wang et al. (2021):** Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*.
- **Virtanen et al. (2020):** SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*.
- **Rudy et al. (2017):** Data-driven discovery of partial differential equations. *Science Advances*.
- **Chen et al. (2021):** Physics-informed learning of governing equations from scarce data. *Nature Communications*.
- **Atkinson et al. (2019):** Data-driven discovery of free-form governing differential equations. *arXiv e-prints*.
- **Jiang & Sun (2025):** Evo-SINDy: Universal Discovery of Partial Differential Equations Using Cooperative Evolutionary Computation. *GECCO '25*.
- **Shojaee et al. (2023):** Transformer-based Planning for Symbolic Regression. *arXiv e-prints*.
- **Raissi et al. (2024):** Physics-Informed Neural Networks and Extensions. *arXiv e-prints*.
- **Raissi et al. (2017):** Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv e-prints*.
- **Cranmer (2023):** Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl. *arXiv e-prints*.
- **Schmidt & Lipson (2009):** Distilling Free-Form Natural Laws from Experimental Data. *Science*.
- **Qian et al. (2022):** D-CODE: Discovering Closed-form ODEs from Observed Trajectories. *ICLR*.
- **Biggio et al. (2021):** Neural Symbolic Regression that Scales. *arXiv e-prints*.
- **Takamoto et al. (2022):** PDEBENCH: An Extensive Benchmark for Scientific Machine Learning. *arXiv e-prints*.
- **Lu et al. (2019):** DeepXDE: A deep learning library for solving differential equations. *arXiv e-prints*.
- **Bongard & Lipson (2007):** Automated reverse engineering of nonlinear dynamical systems. *PNAS*.
- **Fasel et al. (2022):** Ensemble-SINDy: Robust sparse model discovery... *Proceedings of the Royal Society A*.
- **Kaheman et al. (2022):** Automatic differentiation to simultaneously identify nonlinear dynamics... *Machine Learning: Science and Technology*.
- **Xu et al. (2020):** DLGA-PDE: Discovery of PDEs... via combination of deep learning and genetic algorithm. *Journal of Computational Physics*.
- **Maslyaev et al. (2019):** Data-driven PDE discovery with evolutionary approach. *arXiv e-prints*.
- **Kacprzyk et al. (2023):** D-CIPHER: Discovery of Closed-form Partial Differential Equations. *NeurIPS*.
- **Burkardt (2014):** Finite Difference Solution of the Time Dependent 1D Heat Equation... (Online resource).
- **Burkardt (2012):** Finite Difference Method for the 1D Wave Equation (Online resource).
- **Driscoll et al. (2014):** Chebfun Guide. (Book)