Merging Physics and AI

Petr Šimánek

May 16, 2024

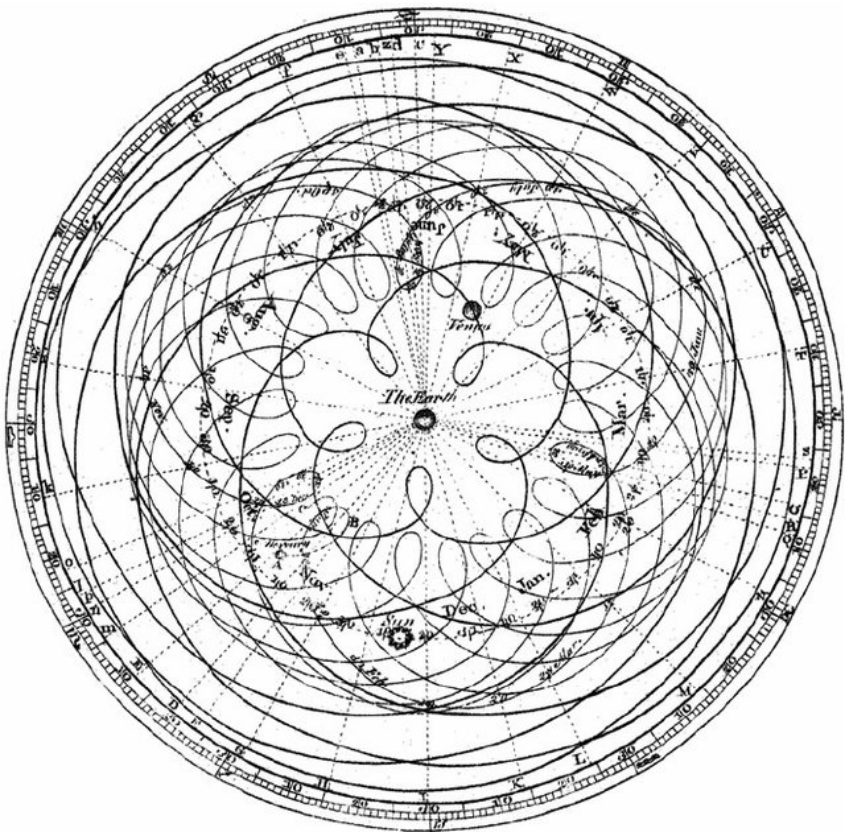
Introduction

- Motivation
- Importance of combining Physics and AI
- SINDy
- Symbolic regression
- Physics-Informed NN (PINN)
- Noether's Theorem and symmetries
- Neural Fourier Operators

Motivation

Geocentric - Ptolemaic system

- Circles-in-circles
- Purely data-driven solution
- Complicated, does not generalize
- Surprisingly accurate!



Motivation

Copernicus heliocentric system - similar to geocentric, but after transformation of data.

- Ellipsis ($\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$)
- Purely data-driven solution
- Much less complicated
- Very accurate!
- Does not explain why, does not generalize

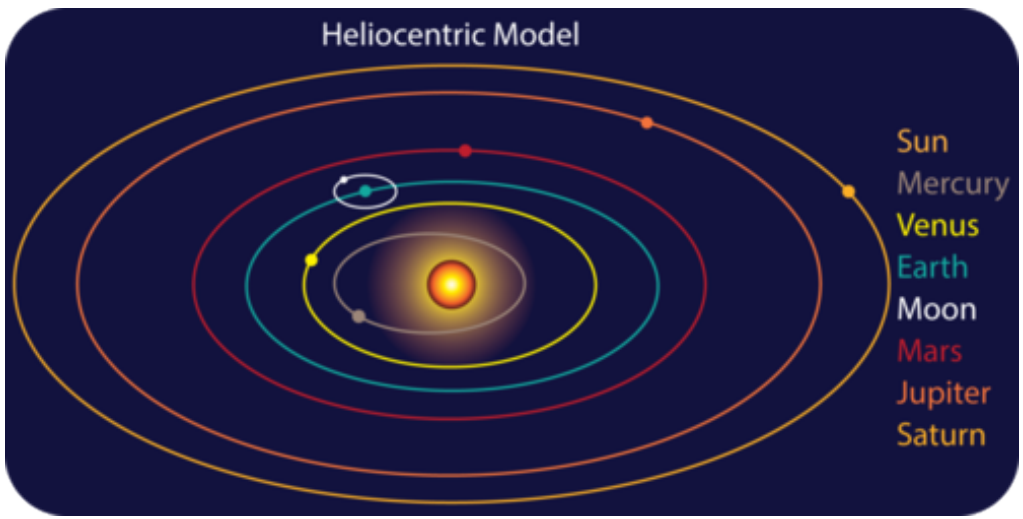


Figure: Heliocentric system

Motivation

Newton's gravitation law

- F=m.G
- Very strong explanatory power
- Generalizes to any system with mass
- Very accurate!

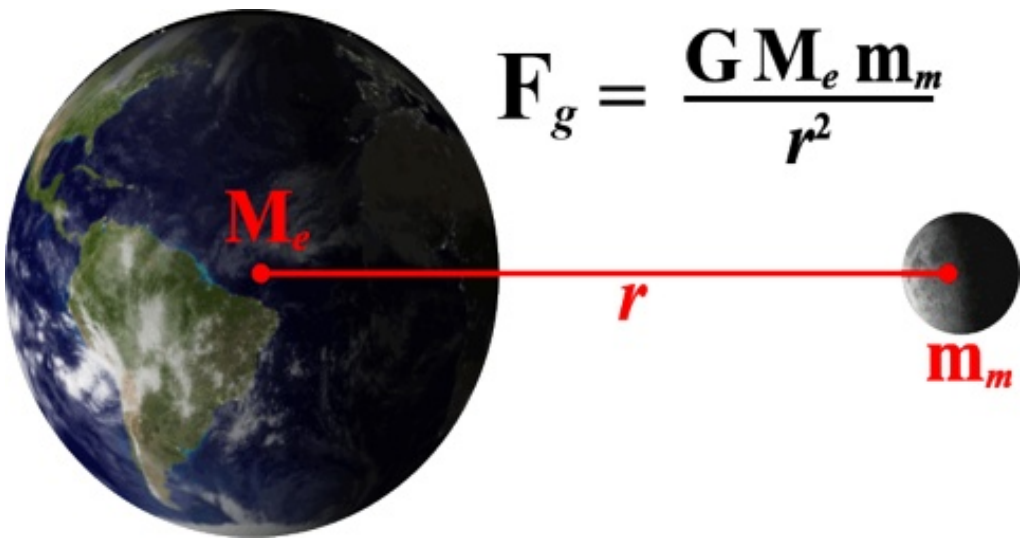


Figure: Newton's gravitational law

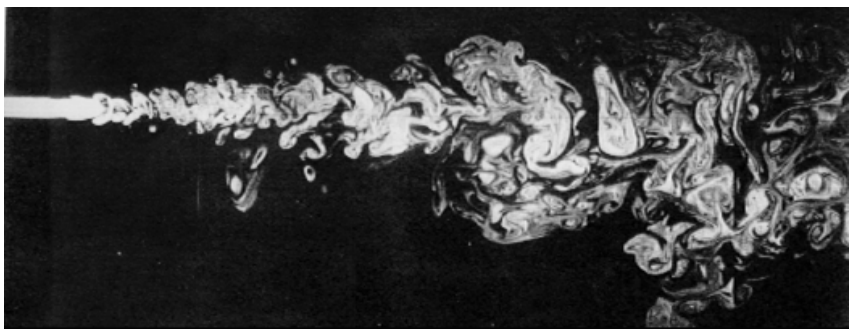
Motivation

What can we do with the measured data?

1. Discover model describing the data (Ptolemaic)
2. Discover the transformation of the data for a simple and well-generalizing model (Copernicus)
3. Discover true governing equations that can be used to explain and understand the model, gives insight (Newton)
4. Solve the equations! (Numerical Mathematics methods)

Background in Physics

- Conservation laws - what is conserved, e.g. momentum, energy?
- Partial Differential Equations (PDEs) - language that describes the physics
- Boundary and initial conditions - what happens out of our computational domain, what was the beginning?



$$
\begin{cases}
\rho \dfrac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla \cdot \boldsymbol{\sigma}(\mathbf{u}, p) = \mathbf{f} & \text{in } \Omega \times (0, T) \\
\nabla \cdot \mathbf{u} = 0 & \text{in } \Omega \times (0, T) \\
\mathbf{u} = \mathbf{g} & \text{on } \Gamma_D \times (0, T) \\
\boldsymbol{\sigma}(\mathbf{u}, p)\hat{\mathbf{n}} = \mathbf{h} & \text{on } \Gamma_N \times (0, T) \\
\mathbf{u}(0) = \mathbf{u}_0 & \text{in } \Omega \times \{0\}
\end{cases}
$$

Figure: Navier-Stokes equations with boundary and initial conditions

How physics can enhance AI models:

- Incorporating physical constraints
- Improving generalization
- Reducing data requirements
- Identifying hidden patterns and relationships
- Accelerating simulations
- Enabling data-driven discoveries
- Enhanced interpretability

Sparse Identification of Nonlinear Dynamics (SINDy)

- SINDy is an algorithm for discovering the governing equations of a dynamical system from data.
- It uses sparse regression to identify the fewest terms (thus governing equations) in a function that can accurately represent the data.
- Formally, given a state measurement matrix X and its derivative $\dot{X}$, SINDy solves the following equation:

$$\dot{X} = \Theta(X)\Xi$$

SINDy: Library of Candidate Functions

- $\Theta(X)$ is a library of candidate functions (e.g., polynomial terms, trigonometric functions) of the state variables.
- Ξ is a sparse vector of coefficients, which we aim to find.
- The sparse regression problem then becomes:

$$\min_{\Xi} ||\dot{X} - \Theta(X)\Xi||_2^2 + \lambda||\Xi||_1$$

where λ is a tuning parameter controlling the sparsity.

SINDy: Algorithm

- The SINDy algorithm works in the following steps:
  1. Construct the library of candidate functions $\Theta(X)$.
  2. Use sparse regression to find Ξ.
  3. Identify the significant terms and discard the rest.

`https://www.youtube.com/watch?v=oqDQwEvHGfE`

SINDy

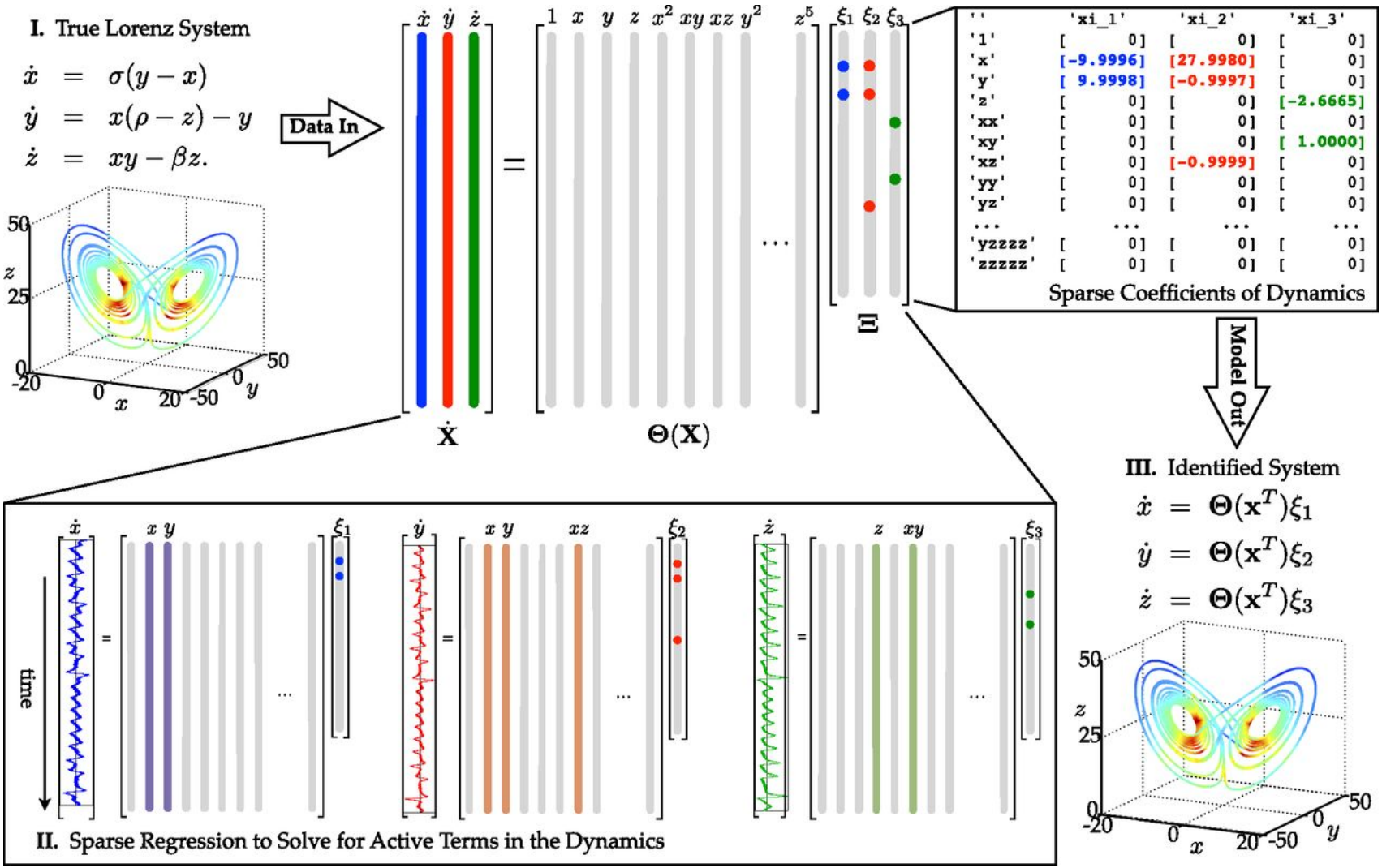


Figure: SINDy

Autoencoder + SINDy

Finding the coordinate system together with the governing equations?
Champion, 2019

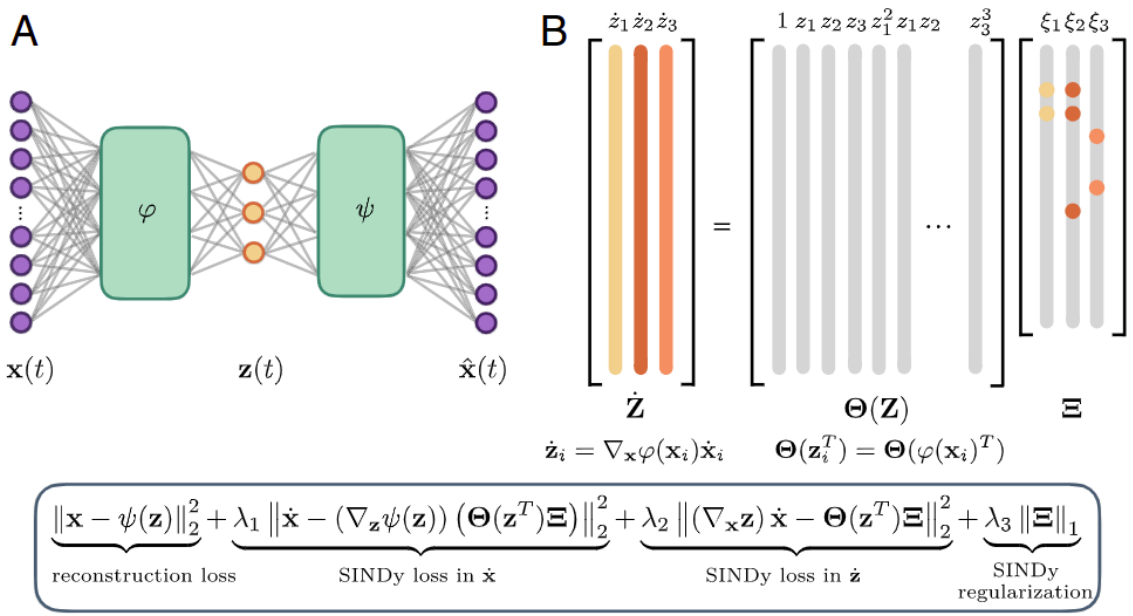


Figure: Autoencoder + SINDy

Uses sequential thresholding to enforce sparsity.

Symbolic Regression: An Overview

- Symbolic regression is an approach in regression analysis that searches for a mathematical equation that best fits a given dataset.
- Unlike traditional regression, which uses a predefined model (like a linear or polynomial model), symbolic regression determines both the form and the parameters of the function.
- This method is particularly powerful because it can uncover underlying patterns and relationships that are not apparent without a predefined model form.
- Symbolic regression often utilizes genetic programming to evolve candidate equations over multiple generations.

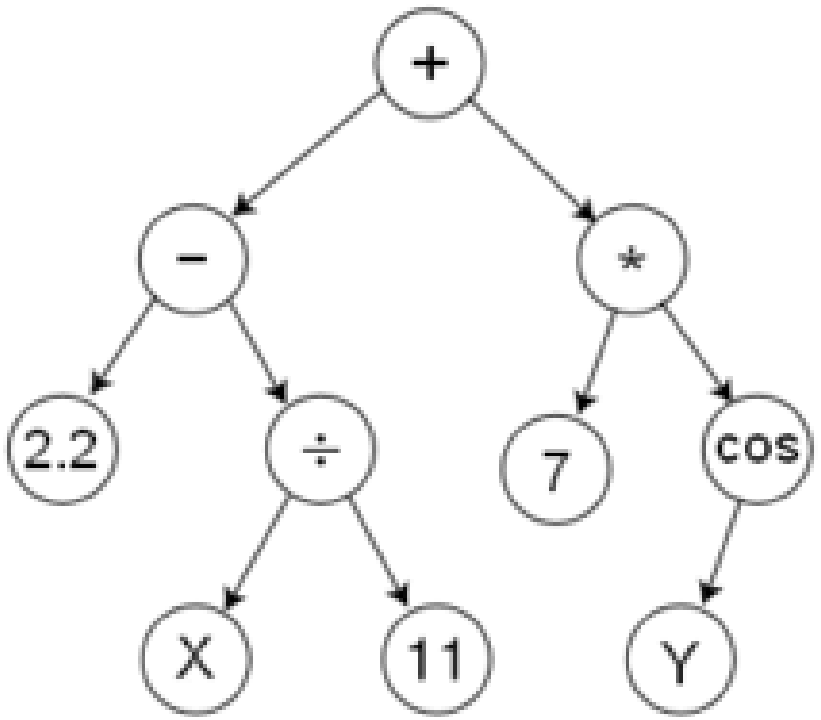
Key Difference

Traditional regression: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$
Symbolic regression: $y = f(x_1, x_2, \ldots, x_n)$, where f is discovered data-driven.

Symbolic Regression via Genetic Programming

- In symbolic regression, GP is used to evolve populations of symbolic expressions to optimally fit the data.
- The GP process involves:
  1. **Initialization:** Generate an initial population of random compositions of functions and terminals.
  2. **Selection:** Select the fittest individuals for reproduction based on their fitness scores.
  3. **Crossover:** Combine parts of two expressions to produce new offspring.
  4. **Mutation:** Randomly alter parts of an expression to create diversity.
  5. **Evaluation:** Compute the fitness of each individual, often using MSE or a similar metric.

Symbolic Regression



$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

Figure: pysr

Challenges in Symbolic Regression

- Symbolic regression is a powerful tool but comes with its own set of challenges:
  - **Overfitting:** Highly expressive models can overfit the data, capturing noise as if it were part of the underlying relationship.
  - **Computational Expense:** Searching through an infinite space of possible models is resource-intensive.
  - **Bloat:** Genetic programming can produce overly complex expressions that are hard to interpret.
  - **Convergence:** Finding a truly optimal symbolic expression is often challenging due to the ruggedness of the fitness landscape.
- PySR is a Python package that is designed to speed up symbolic regression with parallelized operations and by employing strategies to mitigate overfitting and bloat.

Reference

`https://github.com/MilesCranmer/PySR`

Comparing SINDy and Symbolic Regression

Feature	SINDy	Symbolic Regression
Advantages	Fast and efficient for sparse systems; promotes sparsity in models	Highly flexible; can discover unexpected relationships and complex dynamics
Disadvantages	Limited to systems where sparsity is expected; less flexible in functional form	Computationally intensive; can overfit and produce complex models
Applications	Dynamical systems, especially in physics and engineering	Broad range of applications, including empirical formula discovery, feature engineering, and more

Overview of Physics-Informed Neural Networks (PINNs)

- Physics-Informed Neural Networks (PINNs) integrate known physical laws into the learning process of neural networks, enhancing the prediction of systems governed by Partial Differential Equations (PDEs).
- This integration is achieved by embedding the physical equations as part of the loss function, enabling the model to use both data and physics-based constraints to learn.
- PINNs are particularly effective for problems where data is scarce or expensive to obtain but where physical laws are well understood. They are used in fluid dynamics, material science, and more.

Overview of Physics-Informed Neural Networks (PINNs)

Advantages of PINNs

- They offer improved generalization by leveraging physical laws.
- They can solve forward and inverse problems in complex domains.
- They reduce the need for large datasets, unlike conventional deep learning models.

PINNs: Loss Function Structure

- The loss function in PINNs is composed of two primary components:
  - **Data fidelity term** (L_{data}): Encourages the network output to match the observed data points.
  - **Physics-informed term** ($L_{physics}$): Ensures the output satisfies the governing physical equations, typically differential equations.
- The composite loss function is:

$$L = L_{data} + \alpha L_{physics},$$

where α is a tunable parameter that balances the two terms.

PINNs: Loss Function Structure

Formally

If $u_{NN}(x;\theta)$ is the neural network approximation for the true solution $u(x)$, then:

$$L_{data} = \frac{1}{N}\sum_{i=1}^{N} \|u_{NN}(x_i;\theta) - u_i\|^2,$$

$$L_{physics} = \frac{1}{M}\sum_{j=1}^{M} \|F(u_{NN}(x_j;\theta), x_j)\|^2,$$

where F represents the PDE operator applied to the neural network's output.

PINNs: Data Term

- The data term L_{data} measures the discrepancy between the neural network's predictions $u_{NN}(x;\theta)$ and the observed data points $u(x)$, often using squared error for consistency and smooth optimization.
- This term guides the PINN to be accurate on the observed data, anchoring the network's predictions in the empirical reality.

Mathematical Expression

$$L_{data} = \frac{1}{N}\sum_{i=1}^{N} \|u_{NN}(x_i;\theta) - u_i\|^2,$$

where x_i are the points where data is available, and u_i are the observed values.

PINNs: Physics Term

- The physics term $L_{physics}$ enforces the neural network outputs to adhere to the governing physical laws described by differential equations.
- This is done by minimizing the residual of these equations when applied to the network's predictions.

Mathematical Expression

$$L_{physics} = \frac{1}{M} \sum_{j=1}^{M} \|F(u_{NN}(x_j; \theta), x_j)\|^2,$$

where F is the differential operator (representing the PDE) and x_j are collocation points used to evaluate the physics consistency.

PINNs: Computation of PDE Residuals

- The key to enforcing physical laws in PINNs is through the computation of the PDE residuals at selected collocation points.
- Automatic differentiation is utilized to compute derivatives accurately and efficiently, which are crucial for evaluating the PDE residuals.

Procedure

The PDE residual for a general nonlinear PDE, $F(u, \nabla u, \nabla^2 u, \ldots, x) = 0$, is computed as:

$$R(x) = F(u_{NN}(x;\theta), \nabla u_{NN}(x;\theta), \nabla^2 u_{NN}(x;\theta), \ldots, x),$$

where ∇u_{NN} and higher derivatives are obtained via automatic differentiation.

Training Physics-Informed Neural Networks

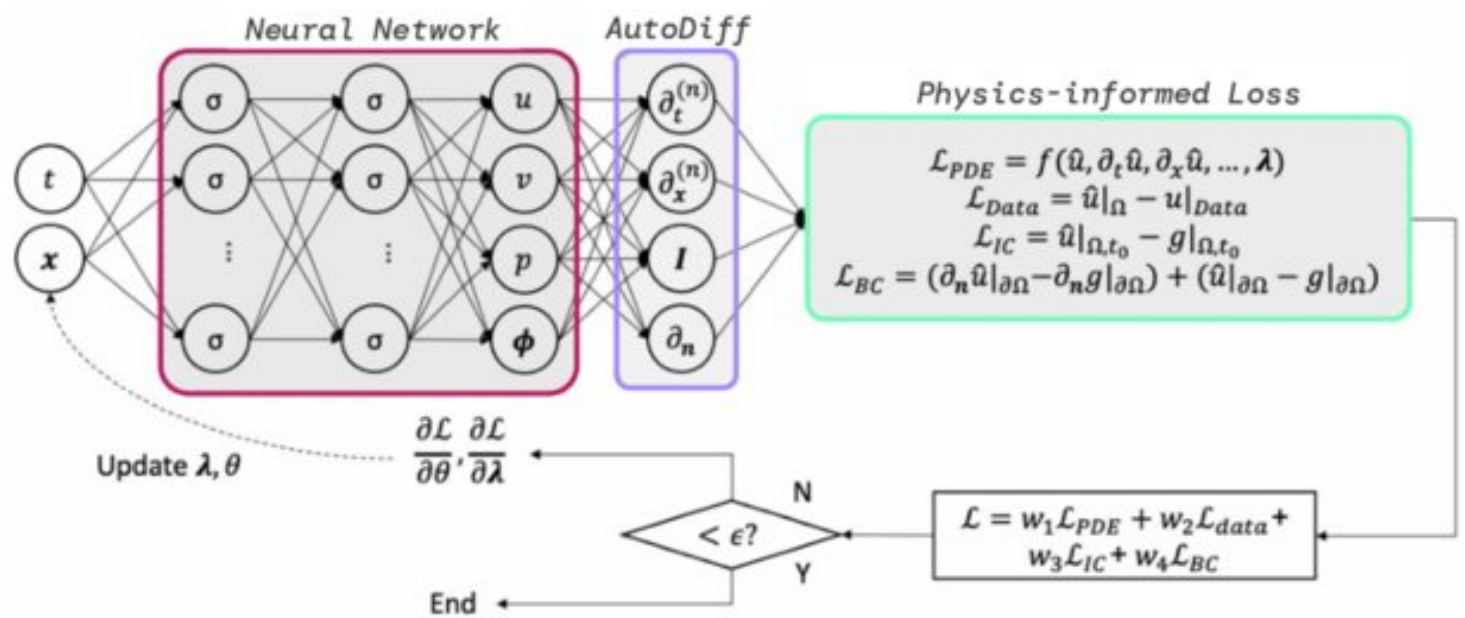


Figure: Schematic of a Physics-Informed Neural Network

Training Physics-Informed Neural Networks

- Training PINNs involves minimizing the composite loss function $L = L_{data} + \alpha L_{physics}$ using gradient-based optimization methods.
- This process adjusts the neural network parameters θ iteratively to improve the model's performance on both data fidelity and physics compliance.

Optimization

Commonly used methods include:

- Stochastic Gradient Descent (SGD) and its variants like Adam and RMSprop.
- L-BFGS, a quasi-Newton method, is often preferred for its efficiency in handling the complexities of the PINNs' loss landscape.

Challenges in Physics-Informed Neural Networks

- While PINNs offer a powerful framework for integrating data and physics, they face several challenges:
  - **Optimization Difficulty:** The non-convexity of the loss surface, combined with the dual nature of the loss function, makes training challenging.
  - **Network Architecture:** The choice of neural network architecture significantly affects the ability of PINNs to learn complex solutions and obey physical laws.
  - **Computational Cost:** PINNs require significant computational resources, especially for high-dimensional problems where the curse of dimensionality can exacerbate training times.
  - **Hyperparameter Tuning:** Balancing the data and physics terms in the loss function (α) is crucial and often requires careful tuning.

Physics-Informed Neural Networks (PINNs)

- Physics-Informed Neural Networks (PINNs) integrate known physical laws into the learning process of neural networks, enhancing the prediction of systems governed by Partial Differential Equations (PDEs).
- This approach can significantly improve generalization and reduce the volume of data required by guiding the network to respect physical laws and constraints.
- Another method to embed physical knowledge is through the use of Noether's theorem, which enforces symmetries and conservation laws in the learning model.

Noether's Theorem and Its Role

- Noether's theorem reveals a profound connection between symmetries and conservation laws in physics, stating that each continuous symmetry corresponds to a conserved quantity.
- For PINNs, this theorem offers a framework to identify and enforce symmetries within the neural network, ensuring the model adheres to physical invariances.
- This approach is crucial for modeling systems where conservation laws derived from symmetries play a significant role, such as in fluid dynamics and quantum mechanics.

Theorem

If a Lagrangian $\mathcal{L}(q, \dot{q}, t)$ has a symmetry under a transformation $q \to q + \delta q$, then:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\delta q\right) = 0,$$

leading to the conservation of the conjugate momentum.

Identifying Symmetries

- Identifying relevant symmetries is a precursor to enforcing them in neural network models.
- Symmetries in physical systems can often be identified by examining the invariance of the action or the Lagrangian under transformations such as translations, rotations, and time shifts.
- Examples include translational symmetry (leading to conservation of linear momentum), rotational symmetry (angular momentum), and temporal symmetry (energy conservation).

Procedure

To identify symmetries:

- Analyze the action $S = \int \mathcal{L}\, dt$ under transformations $x \to x + \delta x$.
- Solve $\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}} \delta x\right) = 0$ to find conserved quantities.

Enforcing Symmetries in Neural Networks

- Enforcing identified symmetries in neural networks ensures that the learned models are physically plausible and adhere to fundamental laws.
- This can be achieved through architectural choices, specialized activation functions, or by modifying the loss function to include symmetry-enforcing penalties.
- Such enforcement leads to more robust and accurate predictive models, especially in physics-driven machine learning.

Methods

- Constraining weights and biases to reflect symmetry properties.
- Using invariant or equivariant layers that respect the symmetry transformations.
- Adding regularization terms to the loss function to penalize symmetry violations.

Constraining Weights and Biases to Enforce Symmetries

- Directly constraining the weights and biases of a neural network is a practical approach to enforce specific symmetries.
- For translation symmetry, this can mean designing networks where the weights are shared across spatial locations (similar to convolutional neural networks).
- For rotational and other symmetries, this could involve more complex constraints or the use of symmetry-adapted layers.

Example

For a rotationally invariant system, one could constrain the weights W by:

$$W = R(\theta) W R(\theta)^{-1},$$

where $R(\theta)$ is a rotation matrix, ensuring that the network's output is invariant under rotations by θ.

Modifying Activation Functions to Enforce Symmetries

- One approach to enforce symmetries in neural networks is by modifying the activation functions to be consistent with the desired symmetries.
- For rotational symmetry, radial basis function (RBF) activations are effective because they respond equally to inputs regardless of the direction, maintaining rotational invariance.
- Similarly, for scale invariance, one might use scale-invariant activations that respond to the ratio of input magnitudes rather than their absolute values.

Example

A Radial Basis Function (RBF) used as an activation function can be defined as:

$$\phi(x) = \exp\left(-\beta \|x - c\|^2\right),$$

where c is the center of the RBF and β controls the spread. This function is rotationally invariant around the center c.

Symmetry-Enforcing Loss Functions

- To further enforce symmetries, symmetry-enforcing terms can be added to the neural network's loss function.
- These terms penalize the network when its outputs violate the specified symmetries, effectively guiding the network towards symmetry-consistent solutions.
- This method is particularly useful for ensuring that learned models respect physical invariances without directly modifying network architecture.

Mathematical Formulation

The symmetry-enforcing loss function can be expressed as:

$$L_{total} = L_{data} + \alpha L_{physics} + \lambda L_{symmetry},$$

where $L_{symmetry}$ measures symmetry violations, and λ is a weight that controls the importance of this term.

Benefits and Challenges of Enforcing Symmetries in Neural Networks

- **Benefits:**
  - **Improved Robustness:** Networks that respect physical symmetries are less prone to spurious predictions and overfitting.
  - **Enhanced Interpretability:** Symmetry-consistent networks often provide solutions that are easier to understand in the context of underlying physical laws.
  - **Increased Physical Consistency:** Enforcing symmetries ensures that predictions adhere to fundamental physical principles, enhancing reliability.

Benefits and Challenges of Enforcing Symmetries in Neural Networks

- **Challenges:**
  - **Complexity in Identification:** Determining the relevant symmetries for a given problem can be non-trivial, especially in complex systems.
  - **Computational Overhead:** Imposing symmetry constraints can increase the computational cost of training neural networks.
  - **Balancing Accuracy and Symmetry:** Finding the right balance between data fidelity and symmetry enforcement requires careful tuning.

Unknown Symmetries

Exploring Unknown Symmetries

When symmetries are not explicitly known, they can be discovered through meta-learning approaches (Allen Zhou, 2021), allowing networks to adaptively learn and enforce these symmetries.

Introduction to Fourier Neural Operator (FNO)

- The Fourier Neural Operator (FNO) is a deep learning architecture designed to learn mappings between function spaces, which are commonly needed in solving complex partial differential equations (PDEs).
- FNO leverages the Fourier transform to parameterize integral kernel operators in a neural network framework, allowing it to efficiently approximate operators by learning in the frequency domain.
- This approach enables FNO to handle high-dimensional PDEs and offers significant computational advantages over traditional and even other neural operator methods due to its use of Fast Fourier Transforms (FFT).

How Does the Fourier Neural Operator Work?

- FNO operates by transforming the inputs into the Fourier domain, where convolution operations used to model interactions are diagonalized and hence more computationally efficient.
- The architecture consists of several layers, each designed to apply a parameterized Fourier integral operator followed by a pointwise nonlinearity.
- The key operation in FNO is the multiplication of the Fourier coefficients of the input functions with learnable complex-valued kernels, making the operation highly parallelizable and efficient.

How Does the Fourier Neural Operator Work?

Mathematical Formulation

Given an input function u, FNO transforms it as:

$$v(x) = \mathcal{F}^{-1}\left(\mathcal{R}(\xi) \cdot \mathcal{F}(u)(\xi)\right),$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier and inverse Fourier transforms, and $\mathcal{R}$ represents the learnable kernel in the frequency domain.

Training and Learning with Fourier Neural Operator

- Training the FNO involves optimizing the parameters of the Fourier kernels $\mathcal{R}(\xi)$ to minimize the difference between the network output and the ground truth for given training data.
- The loss function is typically a norm-based metric in the function space, such as the L^2 norm (MSE).
- Due to the Fourier domain operations, FNO efficiently computes gradients using backpropagation, facilitated by the FFT and its inverse.

Applications of Fourier Neural Operator

- **Fluid Dynamics:** FNO has been effectively used to model flow dynamics in various contexts, including turbulent flows and weather prediction, by learning the Navier-Stokes equations.
- **Materials Science:** In the field of material science, FNO can predict the behavior of complex materials under various conditions by learning the mappings from microscale features to macroscale properties.
- **Climate Modeling:** FNO aids in climate science by providing efficient approximations of large-scale climate models, improving the prediction of temperature, pressure, and other atmospheric variables.
- FNO allows zero-shot super-resolution.

Challenges and Future Directions for Fourier Neural Operator

- **Challenges:**
  - **Handling Non-Periodic Boundaries:** FNO relies on the Fourier transform, which naturally suits periodic or infinite domains. Adapting FNO to non-periodic boundaries remains a challenge.
  - **Multi-Scale Phenomena:** While FNO handles a wide range of scales, extremely multi-scale problems can require careful tuning and adaptation.
  - **Generalization Across Different Equations:** Learning operators that generalize well across different types of PDEs is a complex task that requires further research.

Challenges and Future Directions for Fourier Neural Operator

- **Future Directions:**
  - **Improved Architectures:** Developing more sophisticated FNO architectures to handle a broader range of conditions and PDEs.
  - **Combining with Other Models:** Integrating FNO with other machine learning models, such as reinforcement learning or Bayesian methods, for enhanced predictive power.
  - **Expanding Applications:** Applying FNO to new areas like biomedical engineering, quantum physics, and more.

Thank you!

Thank you!