Merging Physics and AI

Petr Šimánek

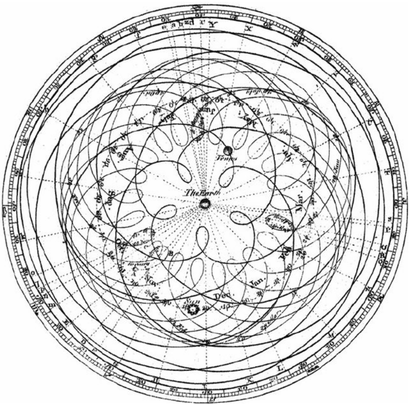
May 22, 2023

Introduction

- Motivation
- Importance of combining Physics and AI
- SINDy
- Symbolic regression
- Physics-Informed Machine Learning (PIML)
- Noether's Theorem and symmetries

Geocentric - Ptolemaic system

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



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

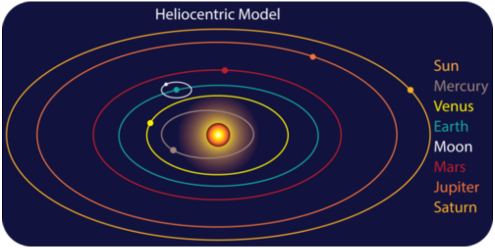


Figure: Heliocentric system

Newton's gravitation law

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

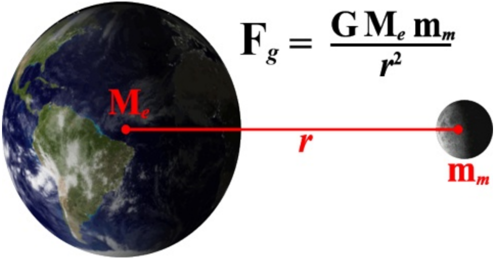


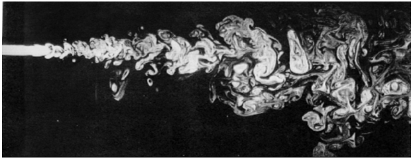
Figure: Newton's gravitational law

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)

- 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

- 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.

- 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

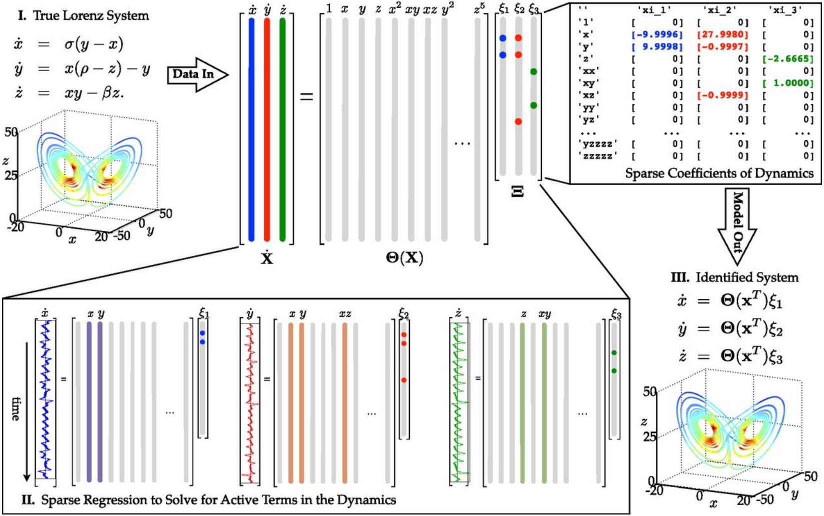


Figure: SINDy

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

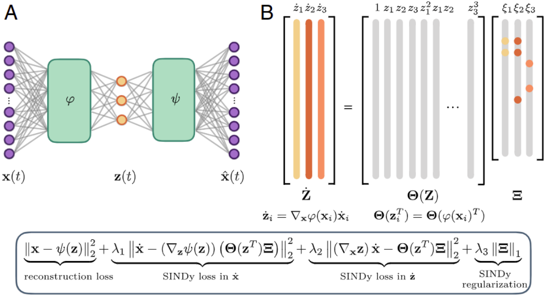


Figure: Autoencoder + SINDy

Uses sequential thresholding to enforce sparsity.

Symbolic Regression

- Symbolic regression is a type of regression analysis that discovers the form of a mathematical equation to best fit a given dataset.
- Unlike traditional regression methods that fit parameters to a pre-defined model, symbolic regression seeks both the form of the function and the numerical parameters that provide the best fit.
- This process is commonly guided by genetic programming.

- Genetic programming (GP) is a method for the automatic induction of computer programs.
- In the context of symbolic regression, GP evolves populations of symbolic expressions to find the one that best fits the data.
- The process consists of initialization, selection, crossover, mutation, and evaluation.

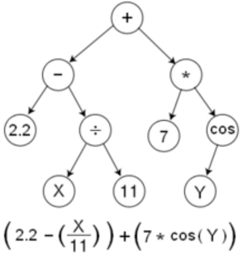


Figure: Symbolic regression

- The fitness function in symbolic regression is often the Mean Squared Error (MSE) between the predicted and actual output values.
- A key advantage of symbolic regression is its ability to produce interpretable models.
- We can interpret the models in the language of mathematics (i.e. no salience maps, no SHAP)

Symbolic Regression: Challenges

- Despite its advantages, symbolic regression faces some challenges:
  - Overfitting: Because symbolic regression can generate very complex models, it risks overfitting the data.
  - Computationally expensive: The search for the best-fitting symbolic model can be time-consuming and computationally expensive.

Current best tool: PySR - highly parallelizable, very fast
`https://github.com/MilesCranmer/PySR`

SINDy vs Symbolic Regression

- Both SINDy and symbolic regression are powerful tools for model discovery from data.
- SINDy is particularly effective for sparse dynamical systems, while symbolic regression provides a more general but computationally expensive approach.
- Both methods complement each other, providing different perspectives on data-driven discovery.

- If we know the equations, we can solve them!
- Applications:
  - Fluid dynamics
  - Materials science
  - Climate modeling
  - Astrophysics
  - LASER behavior
  - Active matter

Using NNs to solve a problem:

$$\mathbf{L}(\mathbf{u}(x,t),\theta) = g$$

$\mathbf{L}$ is some "operator", i.e. some "physics"

Physics-Informed Neural Networks (PINNs)

- PINNs are a class of neural networks that incorporate the governing physical equations (such as PDEs) into the network's loss function.
- By integrating physical constraints, PINNs can improve generalization and reduce data requirements.
- They can be used for a wide range of PDE problems, including time-dependent, non-linear, and high-dimensional ones.

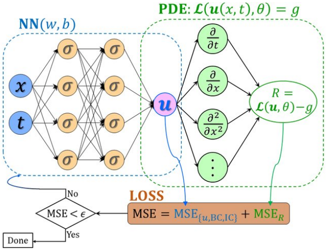


Figure: PINN

- The loss function in PINNs consists of two parts:
  - Data term: This term measures the difference between the network's predictions and the available data.
  - Physics term: This term enforces the underlying physical equations, such as the PDEs.
- The total loss function is a weighted sum of these two terms:

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

where α is a weighting factor.

PINNs: Data Term

- The data term measures the difference between the network's predictions and the available data, typically using a mean squared error (MSE) or another suitable metric.
- This term ensures that the network learns to approximate the observed data points.

PINNs: Physics Term

- The physics term enforces the underlying physical equations (PDEs) on the neural network's predictions.
- This is achieved by computing the residual of the PDE with respect to the neural network's output.
- The residual is then included in the loss function, encouraging the network to satisfy the PDE.

PINNs: Residual Calculation

- To compute the residual, we first differentiate the neural network's output with respect to its inputs.
- Automatic differentiation can be used computation of derivatives.
- The PDE residual is then calculated using these derivatives and the network's output.

PINNs: Training

- PINNs are typically trained using gradient-based optimization methods, such as stochastic gradient descent (SGD) or its variants.

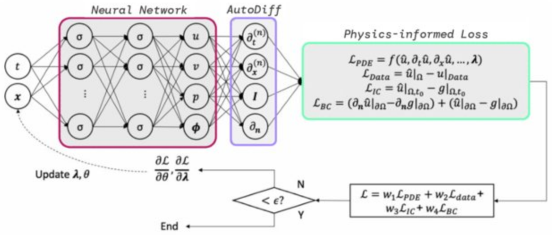


Figure: PINN

- Despite their potential, PINNs face some challenges:
  - Difficulty of optimization: The inclusion of the PDE residual in the loss function can make the optimization problem more complex and harder to solve.
  - Choice of architecture: The architecture of the neural network (number of layers, number of neurons per layer, activation function, etc.) can significantly impact the performance of PINNs.
  - Computational cost: PINNs can be computationally expensive, particularly for complex or high-dimensional problems (i.e. $D > 2$).

PINNs: Future Directions

- PINNs represent a promising direction in the integration of physics and machine learning.
- Future work could focus on developing more efficient training algorithms, designing better network architectures, and extending PINNs to other types of physical systems.

Physics-Informed Neural Networks

- Physics-informed neural networks (PINNs) incorporate physical knowledge into the learning process.
- Other way to include physical knowledge is to use Noether's theorem and enforce symmetries.
- This approach can improve generalization and reduce data requirements.

Noether's Theorem

- Noether's theorem states that for every continuous symmetry of a physical system's action, there is a corresponding conservation law.
- The theorem provides a deep connection between symmetries and conserved quantities in physics.
- It can be used to identify symmetries that should be enforced in a neural network to ensure consistency with physical principles.

- To enforce symmetries in a neural network, we first need to identify the relevant symmetries for the problem at hand.
- These symmetries can be found using Noether's theorem by examining the physical system's action and finding continuous transformations that leave the action invariant.
- Common examples of symmetries include translation (linear momentum), rotation (angular momentum), and time invariance (energy).

Enforcing Symmetries in Neural Networks

- Once the relevant symmetries are identified, we can enforce them in the neural network architecture or learning process.
- This can be done by constraining the weights and biases, modifying the activation functions, or including symmetry-enforcing terms in the loss function.
- Enforcing symmetries helps the network to learn physically consistent solutions.

Constraining Weights and Biases

- Symmetries can be enforced by imposing specific constraints on the weights and biases of the neural network.
- For example, a translation symmetry can be enforced by ensuring that the network's weights are translation-invariant.
- This approach requires careful design of the network architecture and an understanding of how the weights and biases relate to the symmetries.

Modifying Activation Functions

- Symmetries can also be enforced by modifying the activation functions used in the neural network.
- For example, to enforce rotational symmetry, we could use radial basis function (RBF) activation functions, which are invariant to rotations.
- The choice of activation function can have a significant impact on the network's ability to learn and enforce symmetries.

Symmetry-Enforcing Loss Functions

- Symmetries can also be enforced by including symmetry-enforcing terms in the loss function.
- These terms measure the violation of the symmetry by the network's predictions and are added to the usual data-fitting loss term.
- By minimizing this extended loss function, the network learns to make predictions that are consistent with the symmetry.

- Enforcing symmetries can improve the performance of neural networks, making them more robust, interpretable, and physically consistent.
- However, it also presents challenges, such as the difficulty of identifying and enforcing complex symmetries, and the potential increase in computational cost.
- Despite these challenges, the integration of physics and machine learning through methods like symmetry enforcement is a promising direction for future research.

What if the symmetries are not known?
We can learn them through meta-learning (Allen, 2021)

PIML Approaches

- Data-driven discovery: Using ML algorithms to identify patterns and relationships in data, leading to the discovery of governing equations.
- Learning from PDEs: Training ML models to learn solutions of PDEs, incorporating physical constraints and principles into the learning process.
- Hybrid models: Combining physics-based and data-driven models to leverage the strengths of both approaches.

- Neural ODEs are a recent development in deep learning that bridges the gap between neural networks and differential equations.
- They replace the discrete layers of a conventional neural network with a continuous transformation defined by an ordinary differential equation (ODE).
- The ODE is parametrized by a neural network, which allows for learning from data.

Basic Formulation

- The basic formulation of a neural ODE is given by the initial value problem:
- $\frac{dz(t)}{dt} = f(z(t), t, \theta)$, with $z(0) = z_0$
- Here, $z(t)$ is the state at time t, f is a function parametrized by a neural network with parameters θ, and z_0 is the initial state.

Training Neural ODEs

- Neural ODEs are trained by adjusting the parameters θ to minimize a loss function.
- The gradients needed for training are computed using the adjoint method, which is a variant of backpropagation for ODEs.
- This allows for efficient computation of gradients, even for long sequences or high-dimensional state spaces.

- The adjoint method solves an auxiliary ODE backwards in time to compute the gradients.
- This makes the memory cost constant, regardless of the length of the trajectory.
- It is based on the concept of adjoint states in optimal control theory.

Advantages of Neural ODEs

- **Memory Efficiency**: Because gradients are computed with the adjoint method, memory cost is constant regardless of trajectory length.
- **Adaptive Computation**: The ODE solver can adjust its computation steps based on the complexity of the function, leading to potential efficiency gains.
- **Parametric Efficiency**: Neural ODEs use the same function f for all transformations, reducing the number of parameters.
- **Continuous-Time Models**: They are naturally suited for continuous-time data or irregularly sampled data.

Applications of Neural ODEs

- Neural ODEs have been used in a variety of applications, including time series prediction, generative models, and reinforcement learning.
- They are particularly well-suited to problems involving continuous-time or irregularly sampled data.

- **Stochastic Neural ODEs**: Introduce randomness into the dynamics, useful for certain types of data and systems.
- **Controlled Neural ODEs**: Incorporate control inputs into the ODE, useful for reinforcement learning and control problems.
- **Second-Order Neural ODEs**: Use second-order differential equations instead of first-order ones.

Challenges and Future Directions

- While Neural ODEs hold promise, there are still challenges to be addressed, such as ensuring stability and dealing with stiff ODEs.
- Future research directions include developing more efficient solvers, exploring other types of differential equations, and finding new applications.

Thank you!

Thank you!