UNIVERSITÉ DU LUXEMBOURG

Physics Informed Machine Learning

Motivation

The environment is governed by fundamental physical equations, such as the Navier-Stokes equations for fluid dynamics, the heat equation for temperature distribution, and the advection-diffusion equation for pollutant transport, which describe the underlying processes shaping natural systems. When applying machine learning (ML) models to solve environmental data analytics problems, ignoring these physical laws can lead to predictions that violate fundamental principles.

This inconsistency is a general problem in environmental data analytics, as data scarcity is a common challenge, forcing models to extrapolate beyond the training dataset space. This exacerbates the issue of physical infeasibility, making it even more critical to integrate domain knowledge and physical constraints into ML models.

This issue is less prevalent in many ML applications, particularly in domains where data is abundant, the systems being modeled are not governed by strict physical laws, or the focus is on pattern recognition rather than predictive accuracy rooted in real-world constraints. For example, applications like image recognition, natural language processing, or recommendation systems operate in domains where physical principles do not directly constrain the model's outputs, and large datasets typically reduce the need for extrapolation. However, in environmental data analytics, where data scarcity and the need for physical realism are critical, the absence of physics-informed approaches can lead to significant challenges, making this a domain-specific issue.

Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) were introduced in 2017 as a framework to integrate physical information directly into machine learning models.

PINNs are machine learning models that integrate physical intuition into the development of neural networks. PINNs use a neural network to learn a mapping between input and output data, but with an added constraint that the neural network must also '*somehow*' satisfy the underlying physical laws of the system being studied.

PINNs are also known by various other names: physics-guided, physics-embedded, physics-enhanced, physics-constrained, and theory-driven neural networks and machine learning-assisted computational modeling, etc. While these terms may be used interchangeably, there are subtle distinctions in how they emphasize the integration of physical information.

Why Use PINNs?

Neural Networks are famously known for being Universal Approximators: *in a supervised learning context, given enough neurons and adequate data, these networks can learn to model any continuous function regardless of its complexity or non-linearity.*

So why do we need to make thing complex and use PINNs?

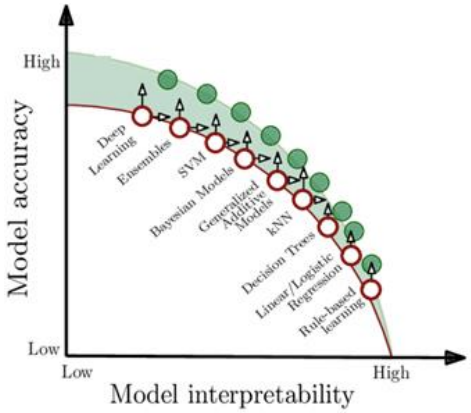
It is well-known that neural networks are constrained by various limitations:

- In supervised learning, they typically require large amounts of data … Sourcing the necessary data is a common challenge,
- They struggle to generalize to parts of the input space that aren't represented in the training dataset,
- Face the challenge of overfitting in presence of noise,
- Are considered black boxes due to the opacity of their functioning,
- May yield outputs that are physically illogical.

By incorporating physical intuition, PINNs strive to address these challenges … although their success is not always guaranteed.

Potential Advantages of PINNs

PINNs incorporate physical intuition as an additional source of information, potentially reducing their reliance on labeled data. By utilizing physical constraints to simplify the problem space, this approach can lead to faster convergence compared to purely data-driven neural networks. Since they use physical intuition, PINNs tend to be more explainable (though not fully).



Model interoperability vs Model accuracy.

The advantages of PINNs are certainly attractive, but it is important to note that:

- ***Necessity of Physical Understanding:*** For successful PINN implementation, a thorough grasp of the system's physics is essential; otherwise, it might fail to learn and generalize effectively. Often, we turn to machine learning models because we lack sufficient knowledge of a problem's underlying physics in the first place.
- ***Significance of Physical Information Integration:*** There are many ways to incorporate physical information into a PINN, and the choice of method can have a significant impact on the network's performance. Sadly, there is no unified or automatable way of finding the best choice.

Why NOT Use PINNs

There are a variety of scenarios in which PINNs may not be the best choice of modeling approach:

- If our understanding of the system is limited, unreliable or unquantifiable,
- If the problem at hand is not physical in nature,
- If the problem is low-dimensional and has a relatively simple alternative model then PINNs are too heavy of a tool to warrant their use.
- If data is too limited While PINNs can handle data-scarce problems, they require some amount of data for training.
- If data is extremely limited, PINNs may not work.
- Limited computational resources PINNs can be computationally expensive to train and may require high-performance computing resources.
- If computational resources are limited, a simpler machine learning approach may be more appropriate.

Representations of Physical Knowledge

When discussing physics in the context of PINNs, there are two primary ways to incorporate physical principles:

1. Equation-Based Representations

- These rely on **strict, well-defined mathematical equations** that are directly derived from fundamental physical laws, such as Newton's laws of motion, Maxwell's equations, or the Navier-Stokes equations.

- They represent a **strong bias** in learning because they tightly constrain the model or solution to adhere to these equations.
- Common forms of these representations include:
    - **Algebraic equations** (e.g., force = mass × acceleration).
    - **Ordinary Differential Equations (ODEs)**, which describe systems evolving over time.
    - **Partial Differential Equations (PDEs)**, which govern spatial and temporal dynamics.
    - **Stochastic Differential Equations (SDEs)**, which include random or probabilistic components.
    - **Integral equations**, which relate functions to their integrals.
- Such representations are precise but can be rigid, requiring significant domain knowledge and computational resources to solve in complex scenarios.

2. Concept-Based Representations

- These focus on **general, qualitative principles of physics** rather than exact equations. They act as broad guidelines rather than strict rules.
- They provide a **weak bias**, offering more flexibility to adapt to data while still respecting basic physical insights.
- Examples of concept-based representations include:
    - **Symmetry constraints**, such as ensuring invariance under rotation or translation, reflecting the fact that physical laws often remain unchanged under such transformations.
    - **Object permanence**, the principle that objects continue to exist even when they are not directly observable, essential in tasks like visual perception or physical reasoning.
- Concept-based approaches are particularly useful in data-driven methods like machine learning, where imposing strict equations may not always be feasible, but embedding general principles can improve model performance and interpretability.

These two approaches—equation-based (strong bias) and concept-based (weak bias)—can also be viewed as complementary.

PINNs and Neural Network Architecture

The concept of PINNs is not tied to a particular neural network architecture Theoretically, any architecture can be adopted to construct PINNs. However, in practice, certain neural network architectures have emerged as more prominent:

- Multi-layer Perceptrons (MLPs) are the most used architecture for PINNs, … due to their simplicity and suitability across a wide range of applications.
- Convolutional neural networks (CNNs) are the go-to choice for developing PINNs involving images, grids, or any spatially structured data.
- Recurrent neural networks (RNNs) are used for modeling time-evolving systems.

Graph neural networks (GNNs) and generative adversarial networks (GANs) are not as commonly employed, their adoption is on the rise.

Training Dynamics in PINNs

PINNs employ the same foundational learning process and optimization algorithms characteristic of conventional data-driven neural networks. PINNs Utilize backpropagation for learning, and often rely on popular optimization algorithms like ADAM, SGD and RMSProp. Nevertheless, Certain methods of integrating physics into neural networks make specific parameters of the network non-trainable.

Incorporating Physics into Neural Networks

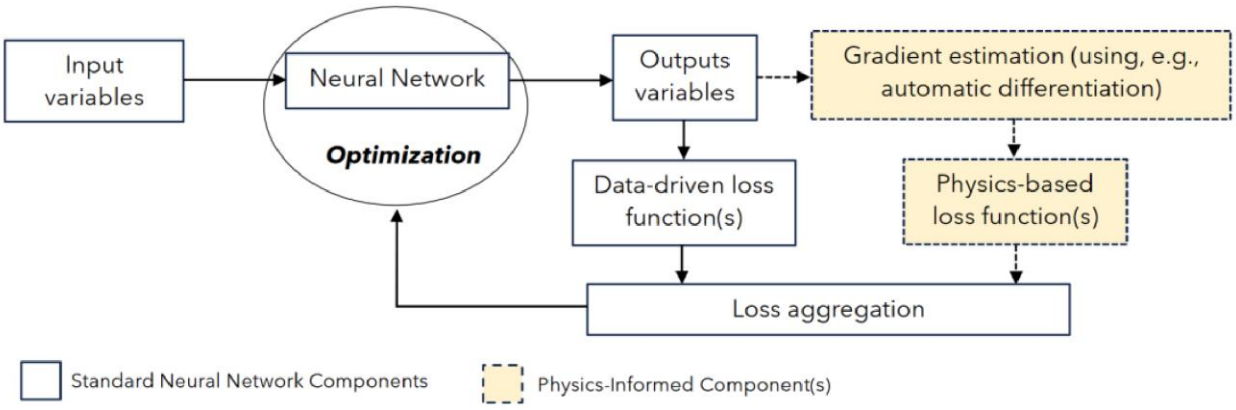
Methods of incorporating physics into neural networks can be classified into three overarching categories:

1. ***Pre-training Physics Integration:*** do not directly affect the neural network's architecture or training process … employed to prepare or manipulate training data.
2. ***In-training Physics Integration:*** infuse physics principles into the neural network's training process.
3. ***Architecture-level Physics Embedding:*** directly modify the neural network's architecture or parameters to explicitly encode physical principles.

Physics-based Loss Function(s)

The most common approach to constructing Physics-Informed Neural Networks (PINNs) involves embedding physical equations directly into the neural network's loss functions. This technique represents a form of *in-training physics integration*.

In this method, the physical equations are reformulated as loss functions that the network seeks to minimize during the training process. This ensures that the learned solution adheres to the underlying physical laws while fitting the data. This loss function measures the discrepancy between the neural network's predictions and the physical constraints defined by the equation. This evaluation is performed at a set of predefined points, known as 'collocation points.'



Schematic of how Physics-based Loss Function are incorporated in PINNs.

Simple Example

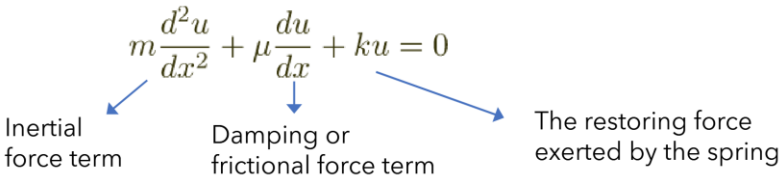
An example involves the motion of a mechanical oscillator or spring pendulum under the influence of a restoring force and friction. While this example is not specific to environmental data analytics, it illustrates a simple concept that can be extended to any physical equation in environmental data analytics. The objective is to estimate the displacement u given the time t and a set of (t, u) measurements.

A Purely Data-driven Approach works by minimizing the difference between the model predictions and the training points by e.g., computing the mean-squared-error:

$$\mathcal{L}_{data-driven} = \frac{1}{N}\sum_{i}^{N}(u_{\mathrm{NN}}(x_i;\theta) - u_{\mathrm{true}}(x_i))^2$$

Where θ represents the learnable parameters in the neural network, and u_{NN} is the displacement prediction by the neural network. The neural network captures the physical process effectively within the range of the experimental data, **But its performance deteriorates when applied beyond this range**. Since the network only relies on the available data, it may not have a complete understanding of the underlying scientific problem.

The physics of the problem can be described by the following ordinary differential equation:



Where m is the mass of the oscillator, μ is the coefficient of friction and k is the spring constant. The idea is to add the residual of the differential equation into the loss function when training the neural network.

The residual for this equation, when using it as a physical loss function in PINNs, is:

$$R(t,u) = m\frac{d^2u_{NN}}{dt^2} + \mu\frac{du_{NN}}{dt} + ku_{NN}$$

The objective in PINNs is to minimize this residual over the entire domain, ensuring that the predictions from the neural network satisfy the physical laws represented by the differential equation.

The physical loss term in the objective function can involve the mean squared error of the residuals in discrete form:

$$\mathcal{L}_{physics-informed} = \frac{1}{M}\sum_{i}^{M}(R_i(t_i,u_i)$$

The loss function is often a weighted sum of the data-driven and physics-informed loss functions:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{data\_driven} + \beta\mathcal{L}_{physics-informed}$$

Incorporating an extra "physics loss" into the loss function aims to guarantee that the network's learned solution is congruent with the established laws of physics.

Estimating the Derivatives in the Physical Loss

Estimating derivatives in the physical loss can be achieved using various methods:

Automatic Differentiation (Autodiff)

- This is by far the most widely used approach for computing derivatives in functions implemented as computer programs.

- It seamlessly integrates with backpropagation, making it highly effective for training PINNs.
- Autodiff is a core feature of most mainstream machine learning libraries, including PyTorch and TensorFlow, enabling efficient and accurate derivative computations.

Other Methods

- **Numerical Differentiation**: Techniques such as finite difference methods can also be employed within PINNs.
- These methods are particularly useful in hybrid approaches, where the strengths of neural networks are combined with traditional numerical techniques to enhance accuracy and robustness.

Physical Regularization

Rather than focusing solely on minimizing physical loss terms, the physical loss can acts as a regularizer to the primary data loss. The main objective remains data fitting, the physics loss ensures the learned function doesn't deviate significantly from physical laws. Done as either soft or hard constraints.

In the soft regularization method, the physics loss is added as a penalty term to the primary data loss:

$$\mathcal{L}_{total} = \mathcal{L}_{data\_driven} + \lambda\, \mathcal{L}_{physics-informed}$$

$\lambda\, \mathcal{L}_{physics-informed}$ → Penalty term

λ → Regularization parameter

A larger λ will enforce the physical laws more strictly.

In the hard regularization method, the optimization problem can be stated as:

$$\min \mathcal{L}_{data-driven}$$

$$Subject\ to\ \mathcal{L}_{physics-informed} = 0$$

Minimized the data-driven loss while ensuring that the predicted function strictly adheres to the physical laws

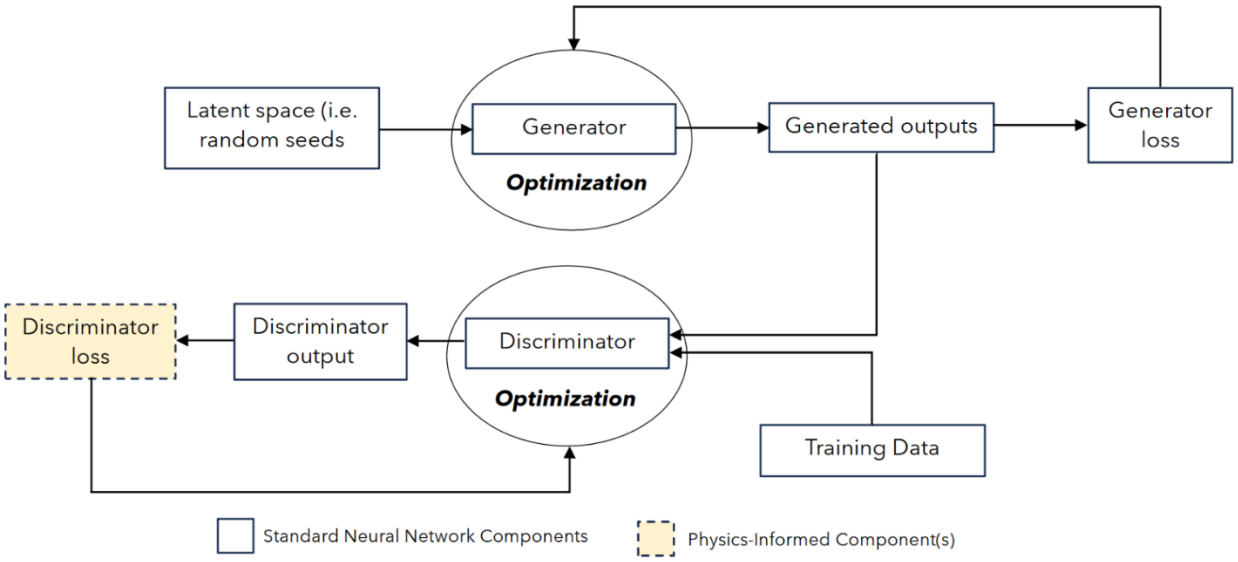
Physics-informed Generative Adversarial Networks

Physics-informed Generative Adversarial Networks (PI-GANs) integrate physical laws into the architecture of Generative Adversarial Networks (GANs) to produce data that is both realistic and adheres to known physical principles.

In the common approach, the generator creates outputs starting from a noise vector or random input. Its objective is to produce data samples that mimic real-world data while implicitly respecting physical constraints.

The discriminator evaluates the outputs from the generator by assessing two main aspects: realism and physical validity. It checks whether the generated data is indistinguishable from real data (realism) and whether it conforms to the governing physical laws or constraints (physical validity). By providing feedback on these aspects, the discriminator guides the generator to improve its outputs over successive iterations.

This process leads the generator to produce data that not only appears realistic but also aligns with physical laws. Incorporating physics-based constraints directly into the discriminator's loss function or using additional physical metrics to evaluate the generated data, ensures that the outputs respect both empirical observations and fundamental physical principles.



Schematic of how Physics-informed Generative Adversarial Networks work.

Architecture-level Physics Embedding

Architecture-Level Physics Embedding involves incorporating physics-based concepts directly into the design of a neural network. These embedded elements can either remain fixed, unaffected by the training process, or be adjusted during training using methods that differ from conventional neural network training approaches.

Example: Physics-Informed Activation Functions

One way to achieve this is by customizing the activation functions of the network to reflect the physics of the problem being modeled.

Mechanical Oscillator Example:

For a system like a mechanical oscillator influenced by restoring and frictional forces, activation functions can be designed to mirror these forces:

1. **Restoring Force Representation** An activation function can mimic the behavior of forces that work to bring a system back to equilibrium, capturing their proportional relationship to displacement.
2. **Frictional Force Representation** To account for friction or damping effects, an activation function can decrease as the input increases, reflecting how friction opposes motion.
3. **Combining Effects** A custom activation function can be designed to integrate both restoring and frictional effects, providing a unified representation of these forces within the network.