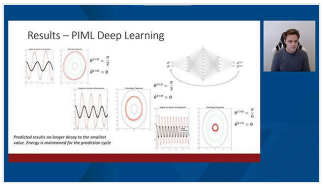
Physics-Informed Neural Networks (PINNs)

What Are Physics-Informed Neural Networks (PINNs)?

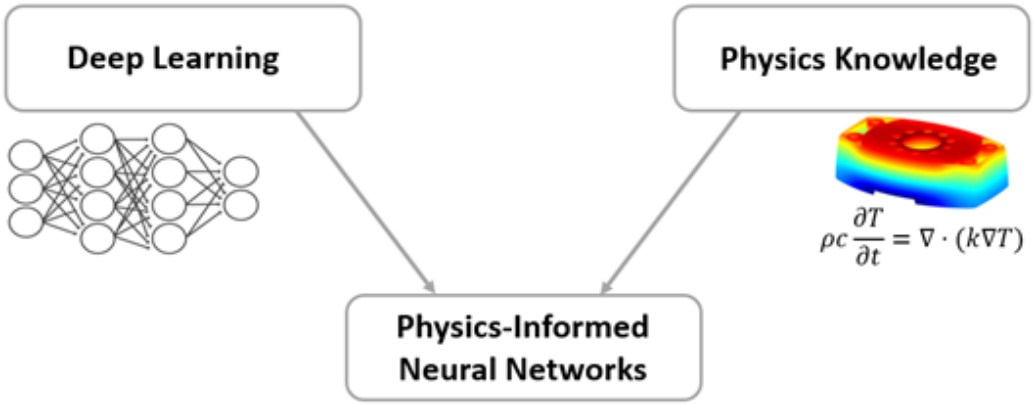


Using Physics-Informed Machine Learning to Improve Predictive Model Accuracy

Physics-informed neural networks (PINNs) are neural networks that incorporate physical laws described by differential equations into their loss functions to guide the learning process toward solutions that are more consistent with the underlying physics. PINNs can be used to:

- Approximate solutions to partial differential equations (PDEs) and ordinary differential equations (ODEs).
- Solve inverse problems, such as estimating model parameters from limited data.

With Deep Learning Toolbox™, you can build and train PINNs, which enable rapid predictive analysis. You can integrate PINNs with MATLAB® and Simulink® for system-level simulation, control design, and design optimization.



Physics-informed neural networks (PINNs) include governing physical laws in the training of deep learning models to enable the prediction and modeling of complex phenomena while encouraging adherence to fundamental physical principles.

The Benefits of PINNs

PINNs are a class of physics-informed machine learning methods that seamlessly integrate physics knowledge with data. Often, PINNs get compared with purely data-driven methods and traditional numerical methods for solving problems involving PDEs and ODEs.

Unlike purely data-driven approaches, which learn mathematical relationships solely from input and output data, PINNs:

- Use prior physics knowledge.
- Make more accurate predictions outside of the training data set.

- Are more effective with limited or noisy training data.

Unlike traditional numerical methods for solving differential equations, such as finite element analysis for PDEs, PINNs:

- Are mesh-free.
- Can approximate high-dimensional PDE solutions.
- Can solve for missing model parameters such as unknown PDE or ODE coefficients.
- Can solve ill-posed problems where no boundary data exists.
- Can easily incorporate sparse or noisy measurements.

While PINNs offer potential benefits compared with purely data-driven methods and traditional numerical methods, PINNs do have some limitations and challenges, including:

- Limited convergence theory
- Lack of unified training strategies
- Computational cost of calculating high-order derivatives
- Difficulty learning high-frequency and multiscale components of PDE solutions

However, PINNs are a dynamic research area, with ongoing advancements expected to address and overcome these current challenges and limitations.

Choosing between PINNs, data-driven approaches, and traditional numerical methods depends on your application. The table below summarizes the benefits and limitations of each method.

	Purely Data-Driven Approaches	Traditional Numerical Methods	PINNs
Incorporate known physics	✗	✓	✓
Generalize well with limited or noisy training data	✗	✗	✓
Solve forward and inverse problems simultaneously	✓	✗	✓
Solve high-dimensional PDEs	✗	✗	✓
Enable fast “online” prediction	✓	✗	✓
Are mesh-free	✓	✗	✓
Have well-understood convergence theory	✗	✓	✗

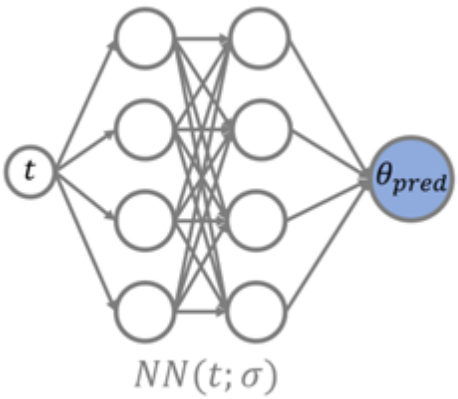
Scale well to high-frequency and multiscale PDEs	✗	✓	✗

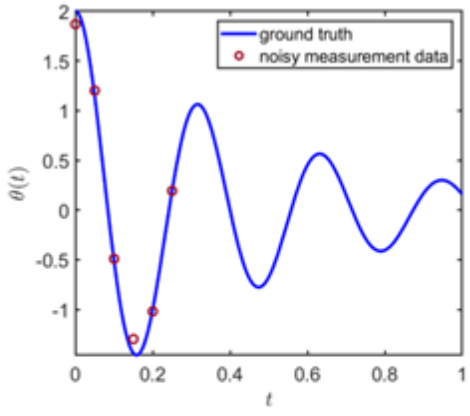
Characteristics of PINNs compared with purely data-driven approaches, which learn mathematical relationships solely from input-output data, and traditional numerical methods, such as FEA for approximating solutions to PDEs.

How PINNs Differ from Traditional Neural Networks

PINNs differ from traditional neural networks in their ability to incorporate a priori domain knowledge of the problem in the form of differential equations. This additional information enables PINNs to make more accurate predictions outside of the given measurement data. Furthermore, the additional physics knowledge regularizes the predicted solution in the presence of noisy measurement data, enabling PINNs to learn the true underlying signal rather than overfitting the noisy data.

For example, consider a scenario where the noisy measurements, θ_{meas}, of a system of interest have been collected, and the objective is to predict future values of the system, θ_{pred}, with a feedforward artificial neural network. The network is trained using the available measurements and will be used to predict unseen future values. Training a regression neural network typically involves minimizing the mean-squared error between the neural network's predictions and the provided measurements.

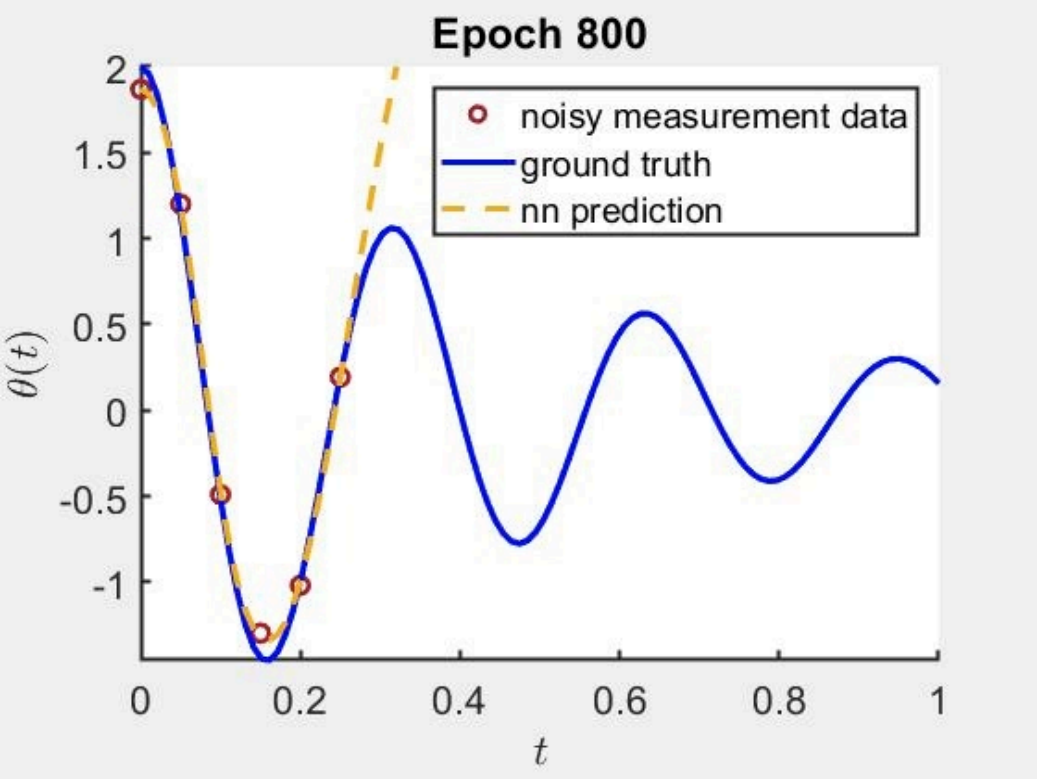




$$\min_{\sigma} \frac{1}{N} \sum_{i=1}^{N} \left| \theta_{pred}(t_i; \sigma) - \theta_{meas}(t_i) \right|^2$$

Traditional neural networks adjust their parameters to minimize the error between the network's prediction and observed measurements.

The neural network struggles to accurately predict values of the system outside of the training data.

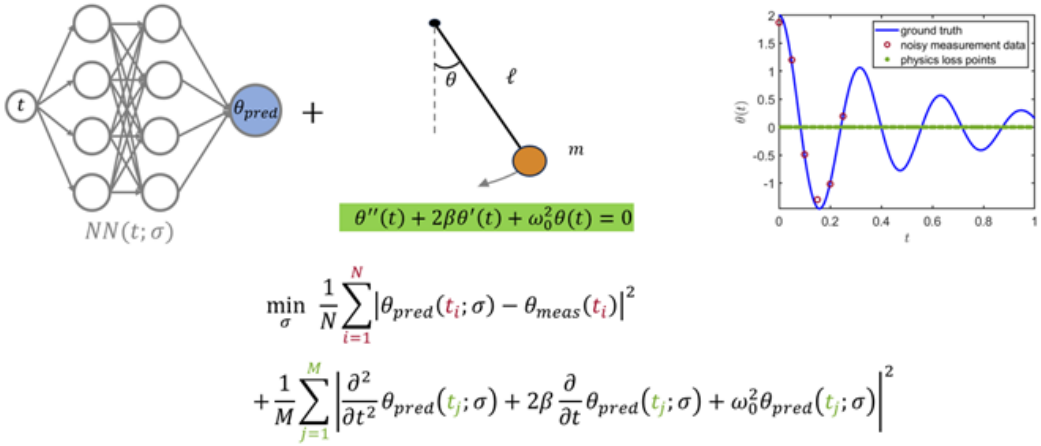


A naïve neural network, trained using the `trainnet` function in Deep Learning Toolbox, overfits noisy measurements and performs poorly for *t* outside of available range. (See MATLAB code.)

Acquiring more data could enhance predictions, yet this approach may be prohibitively expensive or impossible for many applications. However, often the domain expert possesses deeper knowledge about the underlying physical process that governs the system they are studying. Specifically, in this scenario, the measurements represent the angle of displacement from vertical of a payload swinging from a crane. This process can be simplistically represented by a damped pendulum, which can be approximately modeled for small angles by a linear, second-order differential equation:

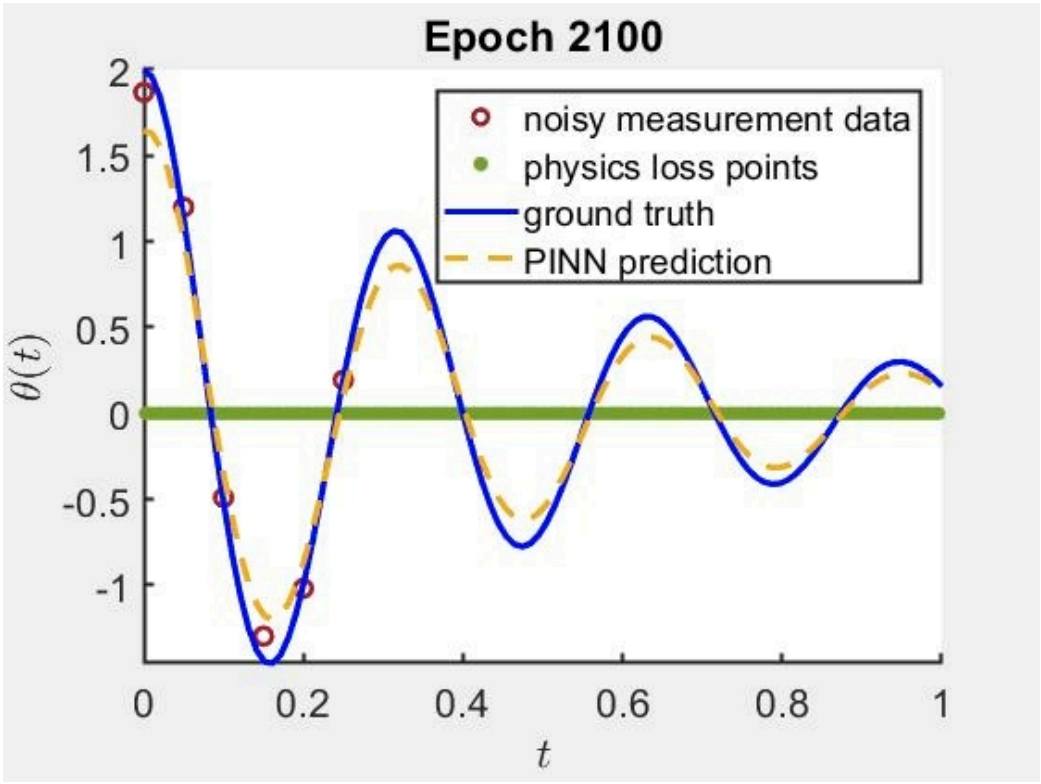
$$\theta''(t) + 2\beta\theta'(t) + \omega_0^2\theta(t) = 0$$

Rather than ignoring this knowledge, PINNs incorporate the differential equation as an additional, physics-informed term in the loss function. PINNs evaluate the residual of the differential equation at additional points in the domain, which provides more information to the PINN without the need for more measurements. While this toy example can be solved analytically, it illustrates the concepts behind PINNs.



PINNs, available in Deep Learning Toolbox, adjust their parameters to balance minimizing the error between the network's prediction and observed measurements and the physics loss.

During training, PINNs find a balance between fitting the given measurements and the underlying physical process.

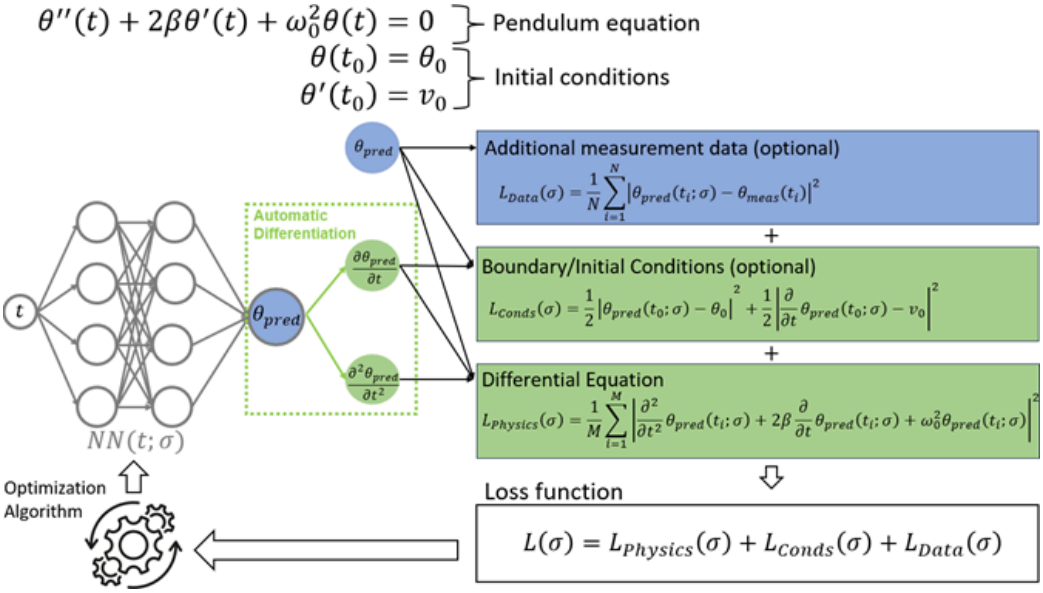


A PINN, created and trained using Deep Learning Toolbox, makes better predictions outside of the measurement data and is more robust to noise than the traditional neural network. (See MATLAB code.)

By incorporating an extra physics loss term, PINNs can outperform traditional neural networks in making predictions in the presence of noisy measurements and in data regimes without measurements.

How PINNs Work

PINNs use optimization algorithms to iteratively update the parameters of a neural network until the value of a specified, physics-informed loss function is decreased to an acceptable level, pushing the network toward a solution of the differential equation.



In training PINNs for an ODE like the pendulum equation, an optimization algorithm adjusts the neural network's parameters to reduce a loss function—comprising the differential equation residual from automatic differentiation (AD), boundary and initial conditions, and optionally other labeled data—to an acceptable level.

PINNs have loss functions, L, which consist of several terms: the physics-informed loss term, $L_{Physics}$, and optionally terms that evaluate the error between the values predicted by the network and any values prescribed by initial and/or boundary data, L_{Conds}, and other additional measurements, L_{Data}. The physics-informed loss term evaluates the residual of the differential equation at points in the domain using automatic differentiation (AD) or other numerical differentiation techniques. Because the physics-informed term does not compute the error between a prediction and a target value, this term can be considered an unsupervised loss term, meaning that the network can be trained with any points from the domain, even without measurements at these points.

First introduced in 2017, PINNs now have many variations, including:

- Bayesian PINNs (BPINNs), which use the Bayesian framework to allow for uncertainty quantification
- Variational PINNs (VPINNs), which incorporate the weak form of a PDE into the loss function
- First-order formulated PINNs (FO-PINNs), which can be faster and more accurate for solving higher-order PDEs than standard PINNs

In addition, PINNs can be used with different neural network architectures, such as graph neural networks (GNNs), Fourier neural operators (FNOs), deep operator networks (DeepONets), and others, yielding so-called physics-informed versions of these architectures.

MATLAB and Deep Learning Toolbox comprehensively support the development of PINNs, from creating or importing diverse neural network architectures, to defining custom physics-informed loss functions with AD, to training using gradient-based optimization algorithms such as ADAM or L-BFGS, and finally to visualizing solutions with advanced MATLAB graphics.

PINN Applications

PINNs leverage the power of deep learning while improving compliance with physical laws, making them a versatile tool for applications where physics is fully or partially known, such as the case of a PDE or ODE with unknown coefficients. Applications of PINNs include:

- **Heat transfer**, specifically for modeling heat distribution and transfer processes. PINNs can embed the governing equations that model thermal processes in materials and systems, such as the heat equation, into the loss function. This approach ensures that the solutions adhere to these physical laws, leading to physically plausible predictions. In addition, PINNs can replace expensive numerical simulations to quickly approximate temperature distributions over parameterized geometries in design optimization applications. Furthermore, PINNs can be used in inverse problems to identify unknown material properties, such as thermal conductivity.
- **Computational fluid dynamics (CFD)**, specifically for approximating velocity, pressure, and temperature fields of fluids by incorporating the Navier-Stokes equations in the loss function. PINNs can be used in mesh-free forward simulations to accurately predict these quantities or in inverse problems where the goal is to infer unknown parameters or inputs, such as boundary conditions, source terms, or fluid properties, from observed data.
- **Structural mechanics**, for solving both forward and inverse problems by embedding the governing physical laws, such as equations of elasticity and structural dynamics, directly into the loss function. This integration enables PINNs to accurately predict structural responses such as deformations, stresses, and strains under various loads and conditions, as well as identify unknown material properties or external loads based on observed data. Particularly useful in scenarios where traditional analytical solutions are infeasible or data is scarce, PINNs reduce the dependency on extensive data sets by leveraging physical principles to guide the learning process. The flexibility of PINNs enables them to handle complex problems, including nonlinear material behavior and multiphysics modeling.

Once created and trained with Deep Learning Toolbox, PINNs can be seamlessly integrated with Optimization Toolbox™ for design optimization, connected to Simulink for system-level simulation, and leveraged across a wide range of other applications.

Examples and How To

- Solve Partial Differential Equation with L-BFGS Method and Deep Learning - Example
- Solve Ordinary Differential Equation Using Neural Network - Example
- Solve Poisson Equation on Unit Disk Using Physics-Informed Neural Networks - Example
- Inverse Problems Using PINNs - Downloadable Code

- Physics-Informed Machine Learning: Cloud-Based Deep Learning and Acoustic Patterning for Organ Cell Growth Research - Article

Software Reference

- `dlgradient`: Compute gradients for custom training loops using automatic differentiation - Function
- `dlnetwork`: Deep learning neural network - Object
- Custom Training Loops - Documentation
- `adamupdate`: Update parameters using adaptive moment estimation (Adam) - Function
- `lbfgsupdate`: Update parameters using limited-memory BFGS (L-BFGS) - Function

See also: Deep Learning Toolbox, Partial Differential Equation Toolbox, finite element analysis, reduced order modeling, Hamiltonian neural network, dynamical system modeling using neural ODE, deep learning, convolutional neural networks (CNNs), generative adversarial networks (GANs), long short-term memory (LSTM) networks, recurrent neural networks (RNNs), neural networks