Physics-informed neural network with unknown measurement noise

Niklas Wahlström,
Joint work with Philipp Pilar

Division of Systems and Control
Department of Information Technology
Uppsala University

April 17, 2024.
Introduction to physics-informed neural networks

Limitations and extensions

Physics-informed neural networks with unknown measurement noise
Introduction to physics-informed neural networks
There are two main strategies to derive and deduce models

- **theory-based** first principles or
- **data-driven** approaches.

My overall research aim is to create new tools for using these two modeling strategies in conjunction.

**Why do we want to do this?**
- Leverage performance of data-driven ML models
- Make ML models more **interpretable**
The combination of machine learning with prior knowledge from physics results in the field of physics-informed machine learning.
Integrating prior knowledge into machine learning models

The combination of machine learning with prior knowledge from physics results in the field of physics-informed machine learning.

- additional term in the loss function

\[ x \rightarrow y \rightarrow \hat{y} \rightarrow \mathcal{L}(y, \hat{y}) + \mathcal{L}_c(\hat{y}) \]
The combination of machine learning with prior knowledge from physics results in the field of physics-informed machine learning.

- additional term in the loss function
- integrated in the model architecture

\[
x \rightarrow \hat{y} \rightarrow \mathcal{L}(y, \hat{y})
\]
The combination of **machine learning** with **prior knowledge** from physics results in the field of **physics-informed machine learning**.

- additional term in the loss function
- integrated in the **model architecture**
- additional **inputs** to the ML model

\[ p \rightarrow x \rightarrow \hat{y} \rightarrow \mathcal{L}(y, \hat{y}) \rightarrow y \]
Integrating prior knowledge into machine learning models

The combination of **machine learning** with **prior knowledge** from physics results in the field of **physics-informed machine learning**.

- **additional term in the loss function**
- integrated in the **model architecture**
- additional **inputs** to the ML model

\[
x \rightarrow \hat{y} \rightarrow \mathcal{L}(y, \hat{y}) + \mathcal{L}_c(\hat{y})
\]
Solving PDEs with machine learning

Damped harmonic oscillator

\[ m \frac{d^2}{dt^2} u + \mu \frac{d}{dt} u + k = 0 \]

Naive approach: Supervised learning

Main problems:

- No guarantee that the model obeys the conservation laws
- May require a lot of training data, not always feasible
Physics-informed neural network

\[ L(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\hat{u}(t_i, \theta) - u_i)^2 + \frac{1}{N_c} \sum_{j=1}^{N_c} \left( \left[ m \frac{d^2}{dt^2} + \mu \frac{d}{dt} + k \right] \hat{u}(t_j, \theta) \right)^2 \]

**From a ML perspective:**

- Physics loss is an unsupervised regularizer, which adds prior knowledge

**From a mathematical perspective:**

- PINNs provide a way to solve PDEs
  - Neural network is a mesh-free functional approximation of PDE solution
  - Physics loss is used to assess if the solution is consistent with PDE
  - Supervised loss is used to include boundary/initial conditions and potential observations
Physics-informed neural networks

PINNs constitute an alternative to classical solvers of PDEs.

\[ \mathbf{\hat{u}} \rightarrow \mathcal{L}_{\text{pde}} + \mathcal{L}_{\text{IC, BC}} + \mathcal{L}_{\text{data}} = \mathcal{L}_{\text{tot}} \]

\[ \mathcal{L}_{\text{pde}} = \frac{1}{N_c} \sum_{j=1}^{N_c} (\mathcal{F}(\mathbf{\hat{u}}(x_j, t_j; \theta))^2 \]

\[ \mathcal{L}_{\text{BC}} = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} (\mathbf{\hat{u}}(x_i, t_i; \theta) - f(x_i, t_i))^2 \]

\[ \mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{k=1}^{N_d} (\mathbf{\hat{u}}(x_k, t_k; \theta) - u_k)^2 \]
PINN training loop

\[
L(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\hat{u}(t_i, \theta) - u_i)^2 + \frac{1}{N_c} \sum_{j=1}^{N_c} \left( \left[ m \frac{d^2}{dt^2} \hat{u} + \mu \frac{d}{dt} \hat{u} + k \right] \hat{u}(t_j, \theta) \right)^2
\]

Training loop

1. Sample boundary/collocation points \( \{t_i\}_{i=1}^{N} \) and \( \{t_j\}_{j=1}^{N_c} \)
2. Compute network outputs
3. Compute \textbf{gradients} of network with respect to \textbf{network input} \( \frac{d}{dt} \hat{u}, \frac{d^2}{dt^2} \hat{u} \)
4. Compute loss
5. Compute \textbf{gradient} of loss function with respect to \textbf{network parameters} \( \frac{d}{d\theta} L \)
6. Take gradient descent step

We can apply autodifferentiation to compute both \( \frac{d}{dt} \hat{u} \) and \( \frac{d}{d\theta} L \)
PINNs are flexible in their use.

\[ \mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{BC}} + \mathcal{L}_{\text{PDE}} \]
PINNs for forward simulation

\[ \mathcal{L}_{\text{PDE}} = \frac{1}{N_c} \sum_{j=1}^{N_c} \left( \nabla^2 - \frac{1}{c(x_j)^2} \frac{\partial^2}{\partial t^2} \right) \hat{u}(x_j, t_j) \right)^2 \]

\[ \mathcal{L}_{\text{IC}} = \frac{1}{N_{\text{IC}}} \sum_{i=1}^{N_{\text{IC}}} (\hat{u}(x_i, t_i) - u_{\text{FD}}(x_i, t_i))^2 \]

Solving the wave equation with physics-informed deep learning – Moseley et al, ArXiv (2020)
PINNs for forward simulation

- Mini-batch size $N_c = N_IC = 500$
- Fully connected NN, 10 layers, 1024 hidden units
- Softplus activation
- Adam optimizer

Training time: 1 hour

Solving the wave equation with physics-informed deep learning – Moseley et al, ArXiv (2020)
PINNs for forward simulation

Ground truth FD

\[ \mathcal{L}_{PDE} = \frac{1}{N_c} \sum_{j=1}^{N_c} \left( \nabla^2 - \frac{1}{c(x_j)^2} \partial^2 \right) \hat{u}(x_j, t_j, s_j) \]

\[ \mathcal{L}_{IC} = \frac{1}{N} \sum_{i=1}^{N} (\hat{u}(x_i, t_i, s_i) - u_{FD}(x_i, t_i, s_i))^2 \]

Conditioned PINNs

Idea: Add IC/BCs as additional network input parameters. Network does not need to be retrained for each simulation ⇒ much faster!

Solving the wave equation with physics-informed deep learning – Moseley et al, ArXiv (2020)
PINNs for inverse problems

\[ \mathcal{L}_{\text{PDE}}(\theta, \phi) = \frac{1}{N_c} \sum_{j=1}^{N_c} \left( \nabla^2 - \frac{1}{c(x_j; \phi)^2} \frac{\partial^2}{\partial t^2} \right) \hat{u}(x_j, t_j; \theta) \right)^2 \]

\[ \mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{i=1}^{N_d} (\hat{u}(x_i, t_i; \theta) - f(x_i, t_i))^2 \]

Treat velocity as another neural network and simultaneously learn it

PINNs for inverse problems

\[ \mathcal{L}_{\text{PDE}}(\theta, \phi) = \frac{1}{N_c} \sum_{j=1}^{N_c} \left( \nabla^2 - \frac{1}{c(x_j; \phi)^2} \frac{\partial^2}{\partial t^2} \right) \hat{u}(x_j, t_j; \theta) \right)^2 \]

\[ \mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left( \hat{u}(x_i, t_i; \theta) - f(x_i, t_i) \right)^2 \]

Treat velocity as another neural network and simultaneously learn it.

Limitations and extensions
Advantages and limitations of PINNs

**Advantages**
- Mesh-free
- Can perform well for high-dimensional PDEs
- Can be extended to inverse problems
- Perform best on messy/mixed problems
  - Noisy data
  - Physics not perfectly known
- Analytical gradients (e.g. sensitivity analysis)

**Disadvantages**
- Computational cost is high
- No guarantee to converge, convergence properties less well understood
- Challenging to scale to more complex problems (larger domains, multi-scale, multi-physics)
Advantages and limitations of PINNs

Advantages

- Mesh-free
- Can perform well for high-dimensional PDEs
- Can be extended to inverse problems
- Perform best on messy/mixed problems
  - Noisy data
  - Physics not perfectly known
- Analytical gradients (e.g. sensitivity analysis)

Disadvantages

- Computational cost is high
- No guarantee to converge, convergence properties less well understood
- Challenging to scale to more complex problems (larger domains, multi-scale, multi-physics)
Hard initial/boundary conditions

**Problem:** How to pick the weight(s)?

\[ \mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{BC}} + \omega \mathcal{L}_{\text{PDE}} \]

\[ \omega \text{ too small } \Rightarrow \text{ only learns boundary conditions} \]

\[ \omega \text{ too large } \Rightarrow \text{ no unique solution} \]

**Example:** Burgers’ equation

\[ \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2} = 0 \quad u(x, 0) = -\sin(\pi x) \]

\[ u(-1, t) = u(1, t) = 0 \]

Let the solution be approximated by

\[ \hat{u}(x, t; \theta) = (x - 1)(x + 1)(t - 0)\text{NN}(x, t; \theta) - \sin(\pi x) \]

Only one loss term to optimize

\[ \mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N_c} \sum_{j=1}^{N_c} \left( \left[ \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2} \right] \hat{u}(x_j, t_j; \theta) \right)^2 \]

Can be challenging to use this approach for complex boundary conditions.

There also exist adaptive schemes for updating the weights.
Adaptive collocation points

**Idea** Place additional collocation points where the PDE residuals are large.

**Algorithm 1** Residual-based adaptive refinement (RAR)

1. Sample the initial collocation points $\mathcal{T}$;
2. Train the PINN for a certain number of iterations;
3. while Training do
   - Sample a set of points $S_0$;
   - Compute the PDE residuals for the points in $S_0$;
   - $S \leftarrow m$ points with largest residuals in $S_0$;
   - $\mathcal{T} \leftarrow \mathcal{T} \cup S$;
   - Train the PINN for a certain number of iterations;
4. end while

Many variations exist, e.g. use PDE residuals to construct a distribution from which new collocation points can be sampled.

Advantages and limitations of PINNs

**Advantages**

- Mesh-free
- Can perform well for **high-dimensional PDEs**
- Can be extended to **inverse problems**
- Perform best on **messy/mixed problems**
  - Noisy data
  - Physics not perfectly known
- Analytical gradients (e.g. sensitivity analysis)

**Disadvantages**

- **Computational cost** is high
- No guarantee to converge, **convergence properties** less well understood
- Challenging to scale to more **complex problems** (larger domains, multi-scale, multi-physics)
Advantages and limitations of PINNs

Advantages

- Mesh-free
- Can perform well for high-dimensional PDEs
- Can be extended to inverse problems
- Perform best on messy/mixed problems
  - Noisy data
  - Physics not perfectly known
- Analytical gradients (e.g. sensitivity analysis)

Disadvantages

- Computational cost is high
- No guarantee to converge, convergence properties less well understood
- Challenging to scale to more complex problems (larger domains, multi-scale, multi-physics)
Physics-informed neural networks with unknown measurement noise
We aim to train PINNs in case of unknown measurement noise.

- **Energy-based model** (EBM) to model unknown noise distribution
- $\mathcal{L}_{\text{data}}^{\text{EBM}}$ ... NLL of measurement-residuals given the learned PDF

\[
\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{data}}^{\text{EBM}}\left(\{y_d - \hat{u}(t_i)\}_{i=1}^{N_d}\right) + \omega \mathcal{L}_{\text{PDE}}(\mathcal{F}, \hat{u}, \{t_j\}_{j=1}^{N_c})
\]

Pilar and Wahlström, Physics-informed neural networks with unknown measurement noise, L4DC (2024)
Example – exponential differential equation

- The solution $u$ is governed by an ODE with unknown $\lambda$.
- The measurements $y$ are contaminated by homogeneous measurement noise $\epsilon$ of unknown form.

\[
\dot{u}(t) = \lambda u(t) \\
y(t) = u(t) + \epsilon
\]

Pilar and Wahlström, Physics-informed neural networks with unknown measurement noise, L4DC (2024)
The **PINN-EBM** also improved the results when considering the **Navier-Stokes equations** in the presence of **non-Gaussian noise**.

\[
f = u_t + \lambda_1(uu_x + vu_y) + p_x - \lambda_2(u_{xx} + u_{yy}) = 0, \\
g = \nu_t + \lambda_1(\nu u_x + \nu v_y) + p_y - \lambda_2(\nu_{xx} + \nu_{yy}) = 0,
\]

---

PINNs constitute an alternative to classical solvers of PDEs.

- Flexible method, both forwards and inverse problems.
- Particularly useful on messy/mixed problems
- Unknown measurement noise can be taken into account in PINNs.

Some pointers if you want to learn more:

- An expert’s guide to training physics-informed neural networks, Wang et al., arXiv (2023)
- Course: ETH Zürich Deep Learning in Scientific Computing (2023) Lectures available on Youtube (some slides heavily inspired from that sources)
Advantages and limitations of PINNs

Advantages

- Mesh-free
- Can perform well for high-dimensional PDEs
- Can be extended to inverse problems
- Perform best on messy/mixed problems
  - Noisy data
  - Physics not perfectly known
- Analytical gradients (e.g. sensitivity analysis)

Disadvantages

- Computational cost is high
- No guarantee to converge, convergence properties less well understood
- Challenging to scale to more complex problems (larger domains, multi-scale, multi-physics)
Advantages and limitations of PINNs

**Advantages**
- Mesh-free
- Can perform well for high-dimensional PDEs
- Can be extended to inverse problems
- Perform best on messy/mixed problems
  - Noisy data
  - Physics not perfectly known
- Analytical gradients (e.g. sensitivity analysis)

**Disadvantages**
- Computational cost is high
- No guarantee to converge, convergence properties less well understood
- Challenging to scale to more complex problems (larger domains, multi-scale, multi-physics)
Random Fourier features

**Problem:** PINNs are biased towards learning low-frequency solutions $\rightarrow$ **spectral bias**

**Idea:** transform inputs to higher-frequency functions $\rightarrow$ **Random Fourier features**

$$
\gamma(x) = \begin{bmatrix}
\cos(Bx) \\
\sin(Bx)
\end{bmatrix}
$$

where $B \in \mathbb{R}^{m \times d}$ with $B_{ij} \sim \mathcal{N}(0, \sigma^2)$ and $\sigma$ is a hyperparameter.

- The coordinate embedding $\gamma(x)$ serves as input to the PINN.
- Enables more effective learning of high frequencies.
- The value of the parameter $\sigma$ is an important design choice.
- In practice, often $\sigma \in [1, 10]$.

Tancik et al. Fourier features let networks learn high frequency functions in low dimensional domains, NeurIPS (2020)