CS598: Physics-Informed Neural Networks: A deep learning framework for solving forward and inverse problems involving nonlinear PDEs

M. Raissi, P. Perdikaris, GE. Karniadakis

Bhavesh Shrimali

November 18, 2021

Table of Contents

Introduction: PINNs

- PINNs - Neural networks that are trained to **solve supervised learning tasks while respecting physical laws (PDEs)**

Introduction: PINNs

- PINNs - Neural networks that are trained to **solve supervised learning tasks while respecting physical laws (PDEs)**
  - Data-driven solution [Raissi et al.(2017a)Raissi, Perdikaris, and Karniadakis]

Introduction: PINNs

- PINNs - Neural networks that are trained to **solve supervised learning tasks while respecting physical laws (PDEs)**
  - Data-driven solution [Raissi et al.(2017a)Raissi, Perdikaris, and Karniadakis]
  - Data-driven discovery of PDEs [Raissi et al.(2017b)Raissi, Perdikaris, and Karniadakis]

Introduction: PINNs

- PINNs - Neural networks that are trained to **solve supervised learning tasks while respecting physical laws (PDEs)**
  - Data-driven solution [Raissi et al.(2017a)Raissi, Perdikaris, and Karniadakis]
  - Data-driven discovery of PDEs [Raissi et al.(2017b)Raissi, Perdikaris, and Karniadakis]
- Two distinct types of algorithms

Introduction: PINNs

- PINNs - Neural networks that are trained to **solve supervised learning tasks while respecting physical laws (PDEs)**
  - Data-driven solution [Raissi et al.(2017a)Raissi, Perdikaris, and Karniadakis]
  - Data-driven discovery of PDEs [Raissi et al.(2017b)Raissi, Perdikaris, and Karniadakis]
- Two distinct types of algorithms
  - New family of *data-efficient* spatio-temporal function approximators

Introduction: PINNs

- PINNs - Neural networks that are trained to **solve supervised learning tasks while respecting physical laws (PDEs)**
  - Data-driven solution [Raissi et al.(2017a)Raissi, Perdikaris, and Karniadakis]
  - Data-driven discovery of PDEs [Raissi et al.(2017b)Raissi, Perdikaris, and Karniadakis]
- Two distinct types of algorithms
  - New family of *data-efficient* spatio-temporal function approximators
  - Arbitrary accurate RK time steppers with potentially unlimited number of stages

Introduction: PINNs

- PINNs - Neural networks that are trained to **solve supervised learning tasks while respecting physical laws (PDEs)**
  - Data-driven solution [Raissi et al.(2017a)Raissi, Perdikaris, and Karniadakis]
  - Data-driven discovery of PDEs [Raissi et al.(2017b)Raissi, Perdikaris, and Karniadakis]
- Two distinct types of algorithms
  - New family of *data-efficient* spatio-temporal function approximators
  - Arbitrary accurate RK time steppers with potentially unlimited number of stages
- Paper: [Raissi et al.(2019)Raissi, Perdikaris, and Karniadakis]

DNNs: Universal Approximators

- Identify a nonlinear map from a few – potentially very high dimensional – input and output pairs of data

PDEs

DNNs: Universal Approximators

- Identify a nonlinear map from a few – potentially very high dimensional – input and output pairs of data
- However, many physical and biological systems consist of **prior knowledge encoded in physical laws**, e.g. Newton's laws of motion, Maxwell's laws of electromagnetism

PDEs

DNNs: Universal Approximators

- Identify a nonlinear map from a few – potentially very high dimensional – input and output pairs of data
- However, many physical and biological systems consist of **prior knowledge encoded in physical laws**, e.g. Newton's laws of motion, Maxwell's laws of electromagnetism
- This prior information can act as **a regularization constraint** that reduces the space of admissible solutions $\implies$ **remove unrealistic solutions that violate fundamental conservation laws (mass, momentum, energy)**

PDEs

DNNs: Universal Approximators

- Identify a nonlinear map from a few – potentially very high dimensional – input and output pairs of data
- However, many physical and biological systems consist of **prior knowledge encoded in physical laws**, e.g. Newton's laws of motion, Maxwell's laws of electromagnetism
- This prior information can act as **a regularization constraint** that reduces the space of admissible solutions $\Longrightarrow$ **remove unrealistic solutions that violate fundamental conservation laws (mass, momentum, energy)**
- Previous ideas employed Gaussian process regression, but these were limited in their ability to handle nonlinear problems

PDEs

DNNs: Universal Approximators

- Identify a nonlinear map from a few – potentially very high dimensional – input and output pairs of data
- However, many physical and biological systems consist of **prior knowledge encoded in physical laws**, e.g. Newton's laws of motion, Maxwell's laws of electromagnetism
- This prior information can act as **a regularization constraint** that reduces the space of admissible solutions $\implies$ **remove unrealistic solutions that violate fundamental conservation laws (mass, momentum, energy)**
- Previous ideas employed Gaussian process regression, but these were limited in their ability to handle nonlinear problems

PDEs

- Need both initial conditions and boundary conditions

DNNs: Universal Approximators

- Identify a nonlinear map from a few – potentially very high dimensional – input and output pairs of data
- However, many physical and biological systems consist of **prior knowledge encoded in physical laws**, e.g. Newton's laws of motion, Maxwell's laws of electromagnetism
- This prior information can act as **a regularization constraint** that reduces the space of admissible solutions $\Longrightarrow$ **remove unrealistic solutions that violate fundamental conservation laws (mass, momentum, energy)**
- Previous ideas employed Gaussian process regression, but these were limited in their ability to handle nonlinear problems

PDEs

- Need both initial conditions and boundary conditions
- Point Collocation methods: Function Approximation + point evaluation, e.g. consider an approximation problem for a function $u(x)$ on $x \in (0,1)$,

$$u(x) \sim \widetilde{u}(x) = a_0 + a_1 x + a_2 x^2; \quad \text{with} \quad \widetilde{u}\left(x_j\right) = \hat{u}_j,\ j = 1, 2$$

Table of Contents

Problem setup

- Parameterized, nonlinear PDE(s)

$$u_t + \mathcal{N}[u; \lambda] = 0, x \in \Omega \subset \mathbb{R}^D,\ t \in [0, T]; \quad (\cdot)_t = \frac{\partial (\cdot)}{\partial t}$$

Problem setup

- Parameterized, nonlinear PDE(s)

$$u_t + \mathcal{N}[u;\lambda] = 0, x \in \Omega \subset \mathbb{R}^D,\ t \in [0, T]; \quad (\cdot)_t = \frac{\partial(\cdot)}{\partial t}$$

- where $u(t, x)$ denotes the latent (hidden) solution, $\mathcal{N}[\cdot;\lambda]$ is a nonlinear operator parametrized by λ

Problem setup

- Parameterized, nonlinear PDE(s)

$$u_t + \mathcal{N}[u;\lambda] = 0, x \in \Omega \subset \mathbb{R}^D,\ t \in [0, T]; \quad (\cdot)_t = \frac{\partial\,(\cdot)}{\partial t}$$

- where $u(t, x)$ denotes the latent (hidden) solution, $\mathcal{N}[\cdot;\lambda]$ is a nonlinear operator parametrized by λ
- The above setup covers a wide range of PDEs in math. physics, including conservation laws, diffusion, reac-diff-advec. PDE, kinetics etc. E.g., Burger's equation in 2D

$$\mathcal{N}[u;\lambda] = \lambda_1 u u_x - \lambda_2 u_{xx} \text{ and } \lambda = (\lambda_1, \lambda_2)\,; \quad (\cdot)_x = \frac{\partial\,(\cdot)}{\partial x} \quad (\cdot)_{xx} = \frac{\partial^2\,(\cdot)}{\partial x^2}$$

Problem setup

- Parameterized, nonlinear PDE(s)

$$u_t + \mathcal{N}[u; \lambda] = 0, x \in \Omega \subset \mathbb{R}^D, \; t \in [0, T]; \quad (\cdot)_t = \frac{\partial (\cdot)}{\partial t}$$

- where $u(t, x)$ denotes the latent (hidden) solution, $\mathcal{N}[\cdot; \lambda]$ is a nonlinear operator parametrized by λ
- The above setup covers a wide range of PDEs in math. physics, including conservation laws, diffusion, reac-diff-advec. PDE, kinetics etc. E.g., Burger's equation in 2D

$$\mathcal{N}[u; \lambda] = \lambda_1 u u_x - \lambda_2 u_{xx} \text{ and } \lambda = (\lambda_1, \lambda_2); \quad (\cdot)_x = \frac{\partial (\cdot)}{\partial x} \quad (\cdot)_{xx} = \frac{\partial^2 (\cdot)}{\partial x^2}$$

- Given λ what is $u(t, x)$ (Inference, filtering and smoothing or simply **data-driven solutions of PDEs**)

Problem setup

- Parameterized, nonlinear PDE(s)

$$u_t + \mathcal{N}[u;\lambda] = 0, x \in \Omega \subset \mathbb{R}^D,\ t \in [0,T]; \quad (\cdot)_t = \frac{\partial(\cdot)}{\partial t}$$

- where $u(t,x)$ denotes the latent (hidden) solution, $\mathcal{N}[\cdot;\lambda]$ is a nonlinear operator parametrized by λ
- The above setup covers a wide range of PDEs in math. physics, including conservation laws, diffusion, reac-diff-advec. PDE, kinetics etc. E.g., Burger's equation in 2D

$$\mathcal{N}[u;\lambda] = \lambda_1 u u_x - \lambda_2 u_{xx} \text{ and } \lambda = (\lambda_1, \lambda_2); \quad (\cdot)_x = \frac{\partial(\cdot)}{\partial x} \quad (\cdot)_{xx} = \frac{\partial^2(\cdot)}{\partial x^2}$$

- Given λ what is $u(t,x)$ (Inference, filtering and smoothing or simply **data-driven solutions of PDEs**)
- Find λ that best describes observations $u(t_i, x_j)$ (Learning, system identification, or **data-driven discovery of PDEs**)

Table of Contents

Data-driven solutions

- Rewrite the PDE as $f(u; t, x) = 0$

$$f(u; t, x) \doteq u_t + \mathcal{N}[u], \quad \text{along with} \quad u = u_\theta\,(t, x)$$

Data-driven solutions

- Rewrite the PDE as $f(u; t, x) = 0$

$$f(u; t, x) \doteq u_t + \mathcal{N}[u], \quad \text{along with} \quad u = u_\theta(t, x)$$

- Along with the above constraint (+ AD) this gives *Physics-informed neural network* **parameterized by θ**

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f$$

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 ; \quad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

Data-driven solutions

- Rewrite the PDE as $f(u; t, x) = 0$

$$f(u; t, x) \doteq u_t + \mathcal{N}[u], \quad \text{along with} \quad u = u_\theta(t, x)$$

- Along with the above constraint (+ AD) this gives *Physics-informed neural network* **parameterized by θ**

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f$$

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 ; \quad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

- $\left\{t_u^i, x_u^i, u^i\right\}_{i=1}^{N_u}$ denote the **initial and boundary training data** on $u(t, x)$

Data-driven solutions

- Rewrite the PDE as $f(u; t, x) = 0$

$$f(u; t, x) \doteq u_t + \mathcal{N}[u], \quad \text{along with} \quad u = u_\theta(t, x)$$

- Along with the above constraint (+ AD) this gives *Physics-informed neural network* **parameterized by θ**

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f$$

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 \ ; \quad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

- $\left\{t_u^i, x_u^i, u^i\right\}_{i=1}^{N_u}$ denote the **initial and boundary training data** on $u(t, x)$
- $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$ specify the **collocation points** for $f(u; t, x)$

Data-driven solutions

- Rewrite the PDE as $f(u; t, x) = 0$

$$f(u; t, x) \doteq u_t + \mathcal{N}[u], \quad \text{along with} \quad u = u_\theta(t, x)$$

- Along with the above constraint (+ AD) this gives *Physics-informed neural network* **parameterized by θ**

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f$$

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 ; \quad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

- $\left\{t_u^i, x_u^i, u^i\right\}_{i=1}^{N_u}$ denote the **initial and boundary training data** on $u(t, x)$
- $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$ specify the **collocation points** for $f(u; t, x)$
- $\mathcal{L}_u$ helps to enforce initial and boundary data accurately, while $\mathcal{L}_f$ imposes the structure of the PDE into the total loss

Table of Contents

Data-driven solutions: Examples

- Most examples that follow have a small number of training data

Data-driven solutions: Examples

- Most examples that follow have a small number of training data
  - Optimizer: L-BFGS (quasi-second order)

Data-driven solutions: Examples

- Most examples that follow have a small number of training data
  - Optimizer: L-BFGS (quasi-second order)
  - Full-batch

Data-driven solutions: Examples

- Most examples that follow have a small number of training data
  - Optimizer: L-BFGS (quasi-second order)
  - Full-batch
- No theoretical guarantees, but as long as the PDE is well-posed $\implies$ optimizer will find the solution

Table of Contents

Schrödinger equation

- Strong form of the PDE (note that $h(t,x) = u(t,x) + \mathrm{i}\, v(t,x)$)

$$
\begin{aligned}
f \doteq ih_t + 0.5h_{xx} + |h|^2 h = 0, \quad & x \in [-5,5], \quad t \in [0,\pi/2] \\
h(0,x) &= 2\,\mathrm{sech}(x) \\
h(t,-5) &= h(t,5) \\
h_x(t,-5) &= h_x(t,5)
\end{aligned}
$$

Schrödinger equation

- Strong form of the PDE (note that $h(t,x) = u(t,x) + \mathrm{i}\, v(t,x)$)

$$
\begin{aligned}
f \doteq ih_t + 0.5h_{xx} + |h|^2 h = 0, \quad & x \in [-5,5], \quad t \in [0, \pi/2] \\
h(0,x) = 2\,\mathrm{sech}(x) & \\
h(t,-5) = h(t,5) & \\
h_x(t,-5) = h_x(t,5) &
\end{aligned}
$$

- Total loss is given as

$$
\mathcal{L} = \underbrace{\mathcal{L}_0 + \mathcal{L}_b}_{=\mathcal{L}_u} + \mathcal{L}_f
$$

Schrödinger equation

- Strong form of the PDE (note that $h(t,x) = u(t,x) + \mathrm{i}\, v(t,x)$)

$$
\begin{aligned}
f \doteq ih_t + 0.5h_{xx} + |h|^2 h = 0, \quad & x \in [-5,5], \quad t \in [0, \pi/2] \\
h(0,x) &= 2\,\mathrm{sech}(x) \\
h(t,-5) &= h(t,5) \\
h_x(t,-5) &= h_x(t,5)
\end{aligned}
$$

- Total loss is given as

$$\mathcal{L} = \underbrace{\mathcal{L}_0 + \mathcal{L}_b}_{=\mathcal{L}_u} + \mathcal{L}_f$$

- Initial/Boundary data:

$$\mathcal{L}_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} \left| h\left(0, x_0^i\right) - h_0^i \right|^2 ; \qquad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

$$\mathcal{L}_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(\left| h^i\left(t_b^i, -5\right) - h^i\left(t_b^i, 5\right) \right|^2 + \left| h_x^i\left(t_b^i, -5\right) - h_x^i\left(t_b^i, 5\right) \right|^2 \right)$$

Schrödinger equation

Training data

Representation

Potential issues

Schrödinger equation

Training data

- Integrate the PDE using a Spectral solver (Chebfun) in space (x)

Representation

Potential issues

Schrödinger equation

Training data

- Integrate the PDE using a Spectral solver (`Chebfun`) in space (x)
- A fourth order explicit RK time-stepper to integrate in time (t) (`scipy.integrate.solve_ivp`)

Representation

Potential issues

Schrödinger equation

Training data

- Integrate the PDE using a Spectral solver (`Chebfun`) in space (x)
- A fourth order explicit RK time-stepper to integrate in time (t) (`scipy.integrate.solve_ivp`)
- $\left\{x_0^i, h_0^i\right\}_{i=1}^{N_0}$ are measurements of $h(t, x)$ at time $t = 0$. Specifically they choose, $N_0 = N_b = 50$ and $N_f = 20,000$

Representation

Potential issues

Training data

- Integrate the PDE using a Spectral solver (`Chebfun`) in space (x)
- A fourth order explicit RK time-stepper to integrate in time (t) (`scipy.integrate.solve_ivp`)
- $\left\{x_0^i, h_0^i\right\}_{i=1}^{N_0}$ are measurements of $h(t, x)$ at time $t = 0$. Specifically they choose, $N_0 = N_b = 50$ and $N_f = 20,000$

Representation

- $h(t, x) = [u(t, x)v(t, x)]$ using a 5-layer deep neural network with 100 neurons per layer

Potential issues

Schrödinger equation

Training data

- Integrate the PDE using a Spectral solver (`Chebfun`) in space (x)
- A fourth order explicit RK time-stepper to integrate in time (t) (`scipy.integrate.solve_ivp`)
- $\left\{x_0^i, h_0^i\right\}_{i=1}^{N_0}$ are measurements of $h(t, x)$ at time $t = 0$. Specifically they choose, $N_0 = N_b = 50$ and $N_f = 20,000$

Representation

- $h(t, x) = [u(t, x)v(t, x)]$ using a 5-layer deep neural network with 100 neurons per layer
- The choice is purely empricial (no theoretical basis (yet)). Bayesian optimization to fine-tune the design of the DNN

Potential issues

Schrödinger equation

Training data

- Integrate the PDE using a Spectral solver (`Chebfun`) in space (x)
- A fourth order explicit RK time-stepper to integrate in time (t) (`scipy.integrate.solve_ivp`)
- $\left\{x_0^i, h_0^i\right\}_{i=1}^{N_0}$ are measurements of $h(t,x)$ at time $t=0$. Specifically they choose, $N_0 = N_b = 50$ and $N_f = 20,000$

Representation

- $h(t,x) = [u(t,x)v(t,x)]$ using a 5-layer deep neural network with 100 neurons per layer
- The choice is purely empricial (no theoretical basis (yet)). Bayesian optimization to fine-tune the design of the DNN

Potential issues

- Continuous time NN models require a large number of collocation points through the domain N_f

Schrödinger equation

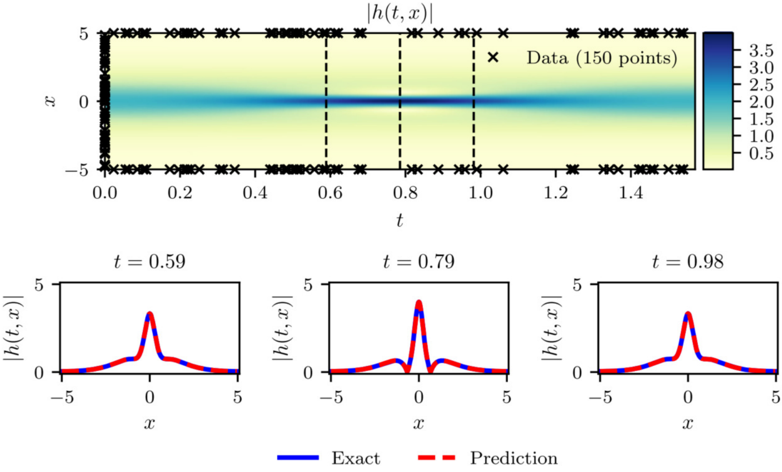


Figure: Top: Boundary and Initial data (150 points), Bottom: Snapshots of the solution of the Schrödinger equation using a PINN

Schrödinger equation

Flexible time-steppers:

Schrödinger equation

Flexible time-steppers:

- Use a generalized RK method with, say, q stages

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}\left[u^{n+c_j}\right], \quad i = 1, \dots, q$$

$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}\left[u^{n+c_j}\right]$$

Schrödinger equation

Flexible time-steppers:

- Use a generalized RK method with, say, q stages

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}\left[u^{n+c_j}\right], \quad i = 1, \ldots, q$$

$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}\left[u^{n+c_j}\right]$$

- The above update can be rewritten as

$$u^n = u_i^n, \quad i = 1, \ldots, q\,; \quad \text{and} \quad u^n = u_{q+1}^n$$

with

$$u_i^n \doteq u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}\left[u^{n+c_j}\right], \quad i = 1, \ldots, q$$

$$u_{q+1}^n \doteq u^{n+1} + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}\left[u^{n+c_j}\right]$$

Table of Contents

Allen-Cahn Equation

Allen-Cahn Equation

- Make use of the above adaptive time-stepper

$$
\begin{aligned}
&u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, \quad x \in [-1, 1], \quad t \in [0, 1],\\
&u(0, x) = x^2 \cos(\pi x),\\
&u(t, -1) = u(t, 1),\\
&u_x(t, -1) = u_x(t, 1).
\end{aligned}
$$

Allen-Cahn Equation

- Make use of the above adaptive time-stepper

$$\begin{aligned}
&u_t - 0.0001 u_{xx} + 5u^3 - 5u = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\
&u(0, x) = x^2 \cos(\pi x), \\
&u(t, -1) = u(t, 1), \\
&u_x(t, -1) = u_x(t, 1).
\end{aligned}$$

- The differential operator: $\mathcal{N}\left[u^{n+c_j}\right] = -0.0001 u_{xx}^{n+c_j} + 5\left(u^{n+c_j}\right)^3 - 5u^{n+c_j}$

Allen-Cahn Equation

- Make use of the above adaptive time-stepper

$$
\begin{aligned}
&u_t - 0.0001 u_{xx} + 5u^3 - 5u = 0, \quad x \in [-1,1], \quad t \in [0,1],\\
&u(0,x) = x^2 \cos(\pi x),\\
&u(t,-1) = u(t,1),\\
&u_x(t,-1) = u_x(t,1).
\end{aligned}
$$

- The differential operator: $\mathcal{N}\left[u^{n+c_j}\right] = -0.0001 u_{xx}^{n+c_j} + 5\left(u^{n+c_j}\right)^3 - 5u^{n+c_j}$
- The loss function is the sum of squared losses

$$
SSE_n = \sum_{j=1}^{q+1} \sum_{i=1}^{N_n} \left| u_j^n \left(x^{n,i}\right) - u^{n,i} \right|^2
$$

$$
\begin{aligned}
SSE_b = &\sum_{i=1}^{q} \left| u^{n+c_i}(-1) - u^{n+c_i}(1) \right|^2 + \left| u^{n+1}(-1) - u^{n+1}(1) \right|^2\\
&+ \sum_{i=1}^{q} \left| u_x^{n+c_i}(-1) - u_x^{n+c_i}(1) \right|^2 + \left| u_x^{n+1}(-1) - u_x^{n+1}(1) \right|^2
\end{aligned}
$$

Allen-Cahn Equation

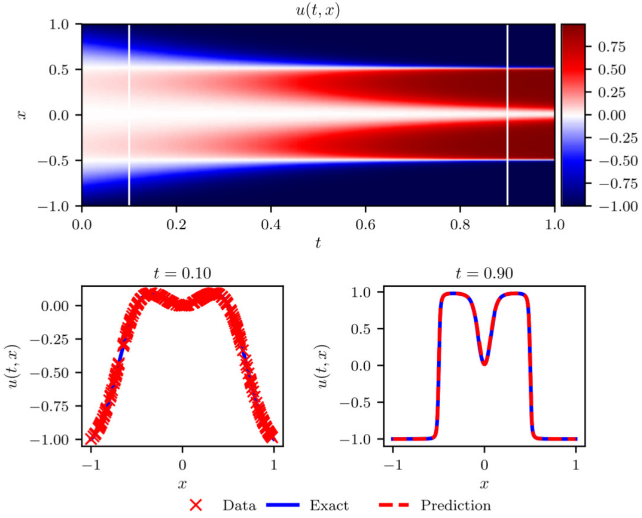


Table of Contents

Data-driven discovery of PDEs: Navier Stokes

Navier-Stokes Equations

Navier-Stokes Equations

- Describe the physics of many phenomena, such as weather, ocean currents, water flow in a pipe and air flow around a wing.

$$\begin{aligned} u_t + \lambda_1 (u u_x + v u_y) &= -p_x + \lambda_2 (u_{xx} + u_{yy}) \\ v_t + \lambda_1 (u v_x + v v_y) &= -p_y + \lambda_2 (v_{xx} + v_{yy}) \end{aligned}; \quad \text{where} \quad (\cdot)_x = \frac{\partial (\cdot)}{\partial x}$$

Data-driven discovery of PDEs: Navier Stokes

Navier-Stokes Equations

- Describe the physics of many phenomena, such as weather, ocean currents, water flow in a pipe and air flow around a wing.

$$\begin{array}{c} u_t + \lambda_1 (u u_x + v u_y) = -p_x + \lambda_2 (u_{xx} + u_{yy}) \\ v_t + \lambda_1 (u v_x + v v_y) = -p_y + \lambda_2 (v_{xx} + v_{yy}) \end{array}; \quad \text{where} \quad (\cdot)_x = \frac{\partial (\cdot)}{\partial x}$$

- $u(t, x, y)$ denotes the x-component of the velocity, $v(t, x, y)$ denotes the y component and $p(t, x, y)$ the pressure

Data-driven discovery of PDEs: Navier Stokes

Navier-Stokes Equations

- Describe the physics of many phenomena, such as weather, ocean currents, water flow in a pipe and air flow around a wing.

$$\begin{array}{l} u_t + \lambda_1 (u u_x + v u_y) = -p_x + \lambda_2 (u_{xx} + u_{yy}) \\ v_t + \lambda_1 (u v_x + v v_y) = -p_y + \lambda_2 (v_{xx} + v_{yy}) \end{array}; \quad \text{where} \quad (\cdot)_x = \frac{\partial (\cdot)}{\partial x}$$

- $u(t, x, y)$ denotes the x-component of the velocity, $v(t, x, y)$ denotes the y component and $p(t, x, y)$ the pressure
- Conservation of mass: $u_x + v_y = 0 \implies u = \psi_y, \quad v = -\psi_x$

Navier-Stokes Equations

- Describe the physics of many phenomena, such as weather, ocean currents, water flow in a pipe and air flow around a wing.

$$\begin{aligned} u_t + \lambda_1 \left(u u_x + v u_y\right) &= -p_x + \lambda_2 \left(u_{xx} + u_{yy}\right) \\ v_t + \lambda_1 \left(u v_x + v v_y\right) &= -p_y + \lambda_2 \left(v_{xx} + v_{yy}\right) \end{aligned}; \quad \text{where} \quad (\cdot)_x = \frac{\partial (\cdot)}{\partial x}$$

- $u(t, x, y)$ denotes the x-component of the velocity, $v(t, x, y)$ denotes the y component and $p(t, x, y)$ the pressure
- Conservation of mass: $u_x + v_y = 0 \implies u = \psi_y, \quad v = -\psi_x$
- Given a set of observations: $\left\{t^i, x^i, y^i, u^i, v^i\right\}_{i=1}^{N}$

$$\begin{aligned} f &\doteq u_t + \lambda_1 \left(u u_x + v u_y\right) + p_x - \lambda_2 \left(u_{xx} + u_{yy}\right) \\ g &\doteq v_t + \lambda_1 \left(u v_x + v v_y\right) + p_y - \lambda_2 \left(v_{xx} + v_{yy}\right) \end{aligned}$$

Data-driven discovery of PDEs: Navier Stokes

Navier-Stokes Equations

- Describe the physics of many phenomena, such as weather, ocean currents, water flow in a pipe and air flow around a wing.

$$\begin{aligned} u_t + \lambda_1 (u u_x + v u_y) &= -p_x + \lambda_2 (u_{xx} + u_{yy}) \\ v_t + \lambda_1 (u v_x + v v_y) &= -p_y + \lambda_2 (v_{xx} + v_{yy}) \end{aligned}; \quad \text{where} \quad (\cdot)_x = \frac{\partial(\cdot)}{\partial x}$$

- $u(t, x, y)$ denotes the x-component of the velocity, $v(t, x, y)$ denotes the y component and $p(t, x, y)$ the pressure
- Conservation of mass: $u_x + v_y = 0 \implies u = \psi_y, \quad v = -\psi_x$
- Given a set of observations: $\{t^i, x^i, y^i, u^i, v^i\}_{i=1}^N$

$$\begin{aligned} f &\doteq u_t + \lambda_1 (u u_x + v u_y) + p_x - \lambda_2 (u_{xx} + u_{yy}) \\ g &\doteq v_t + \lambda_1 (u v_x + v v_y) + p_y - \lambda_2 (v_{xx} + v_{yy}) \end{aligned}$$

- Learn $\lambda = \{\lambda_1, \lambda_2\}$, and pressure field $p(t, x, y)$ by jointly approximating $[\psi(t, x, y) \quad p(t, x, y)]$ with a single NN with two outputs

Navier-Stokes Equations

Navier-Stokes Equations

- Train by minimizing the total loss

$$
\begin{aligned}
\mathcal{L} \doteq\ & \frac{1}{N}\sum_{i=1}^{N}\left(\left|u\left(t^i, x^i, y^i\right) - u^i\right|^2 + \left|v\left(t^i, x^i, y^i\right) - v^i\right|^2\right) \\
& + \frac{1}{N}\sum_{i=1}^{N}\left(\left|f\left(t^i, x^i, y^i\right)\right|^2 + \left|g\left(t^i, x^i, y^i\right)\right|^2\right)
\end{aligned}
$$

Data-driven discovery of PDEs: Navier Stokes

Navier-Stokes Equations

- Train by minimizing the total loss

$$\mathcal{L} \doteq \frac{1}{N}\sum_{i=1}^{N}\left(\left|u\left(t^i,x^i,y^i\right)-u^i\right|^2+\left|v\left(t^i,x^i,y^i\right)-v^i\right|^2\right)$$
$$+\frac{1}{N}\sum_{i=1}^{N}\left(\left|f\left(t^i,x^i,y^i\right)\right|^2+\left|g\left(t^i,x^i,y^i\right)\right|^2\right)$$

- Consider the prototypical problem of an incompressible flow past a (rigid) cylinder, with various values of the Reynolds number $Re = u_\infty D/\nu$

Data-driven discovery of PDEs: Navier Stokes

Navier-Stokes Equations

- Train by minimizing the total loss

$$\mathcal{L} \doteq \frac{1}{N}\sum_{i=1}^{N}\left(\left|u\left(t^i,x^i,y^i\right)-u^i\right|^2+\left|v\left(t^i,x^i,y^i\right)-v^i\right|^2\right)$$
$$+\frac{1}{N}\sum_{i=1}^{N}\left(\left|f\left(t^i,x^i,y^i\right)\right|^2+\left|g\left(t^i,x^i,y^i\right)\right|^2\right)$$

- Consider the prototypical problem of an incompressible flow past a (rigid) cylinder, with various values of the Reynolds number $Re = u_\infty D/\nu$
- Training data is generated using a high-res spectral solver (`NekTar`)

Data-driven discovery of PDEs: Navier Stokes

Navier-Stokes Equations

- Train by minimizing the total loss

$$
\begin{aligned}
\mathcal{L} \doteq & \frac{1}{N}\sum_{i=1}^{N}\left(\left|u\left(t^i,x^i,y^i\right)-u^i\right|^2+\left|v\left(t^i,x^i,y^i\right)-v^i\right|^2\right) \\
& + \frac{1}{N}\sum_{i=1}^{N}\left(\left|f\left(t^i,x^i,y^i\right)\right|^2+\left|g\left(t^i,x^i,y^i\right)\right|^2\right)
\end{aligned}
$$

- Consider the prototypical problem of an incompressible flow past a (rigid) cylinder, with various values of the Reynolds number $Re = u_\infty D/\nu$
- Training data is generated using a high-res spectral solver (`NekTar`)
- Higher order piecewise approximation in space (tenth-order jacobi polynomials), third-order approximation in time (stable for stiff problems)

Navier-Stokes Equations

- Train by minimizing the total loss

$$\mathcal{L} \doteq \frac{1}{N}\sum_{i=1}^{N}\left(\left|u\left(t^i,x^i,y^i\right)-u^i\right|^2+\left|v\left(t^i,x^i,y^i\right)-v^i\right|^2\right)$$
$$+\frac{1}{N}\sum_{i=1}^{N}\left(\left|f\left(t^i,x^i,y^i\right)\right|^2+\left|g\left(t^i,x^i,y^i\right)\right|^2\right)$$

- Consider the prototypical problem of an incompressible flow past a (rigid) cylinder, with various values of the Reynolds number $Re = u_\infty D/\nu$
- Training data is generated using a high-res spectral solver (`NekTar`)
- Higher order piecewise approximation in space (tenth-order jacobi polynomials), third-order approximation in time (stable for stiff problems)
- Given stream-wise $u(t,x,y)$ and transverse $v(t,x,y)$ velocity data, identify unknown $\lambda = \{\lambda_1, \lambda_2\}$ as well as reconstruct $p(t,x,y)$

Navier-Stokes PDE

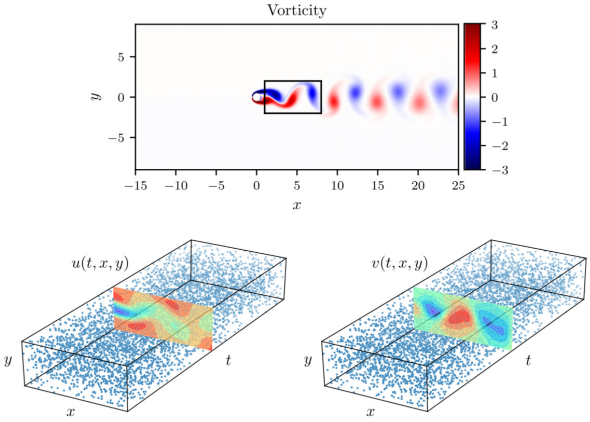


Figure: Navier-Stokes equation: **Top**: Incompressible flow and dynamic vortex shedding past a circular cylinder at Re $= 100$. The spatio-temporal training data correspond to the depicted rectangular region in the cylinder wake. **Bottom**: Locations of training data-points for the stream-wise and transverse velocity components, $u(t, x, y)$ and $v(t, x, t)$, respectively.

Navier-Stokes PDE: Observations

- Set $N = 5000 \sim 1\%$ of the total available data

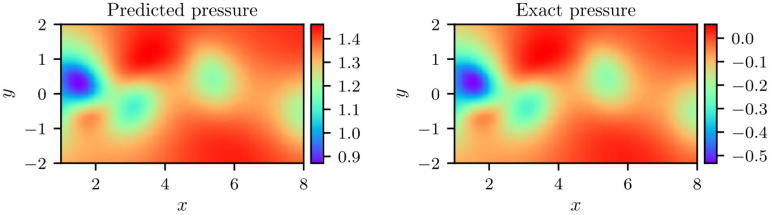


Figure: Results for predicted pressure field

Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01\,(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01\,(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.999\,(uu_x + vu_y) = -p_x + 0.01047\,(u_{xx} + u_{yy})$ $v_t + 0.999\,(uv_x + vv_y) = -p_y + 0.01047\,(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.998\,(uu_x + vu_y) = -p_x + 0.01057\,(u_{xx} + u_{yy})$ $v_t + 0.998\,(uv_x + vv_y) = -p_y + 0.01057\,(v_{xx} + v_{yy})$

Table: Correct partial differential equation along with the identified one obtained by learning λ_1, λ_2 and $p(t, x, y)$.

Table of Contents

Korteweg-de Vries equation

Korteweg-de Vries equation

- KdV equation has higher order derivatives (models shallow water waves)

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xxx} = 0$$

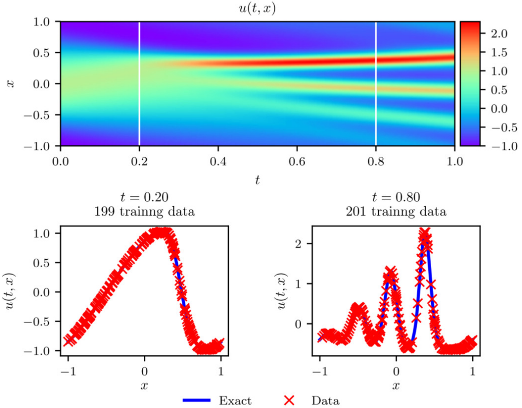
Korteweg-de Vries equation

- KdV equation has higher order derivatives (models shallow water waves)

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xxx} = 0$$

- Learn a set of parameters (similar to NS)

$$\mathcal{N}\left[u^{n+c_j}\right] = \lambda_1 u^{n+c_j} u_x^{n+c_j} - \lambda_2 u_{xxx}^{n+c_j}$$



Korteweg-de Vries equation

Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Remarks

Korteweg-de Vries equation

Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Remarks

- Traditional spectral solvers require a smaller time-increment to achieve similar accuracy

Korteweg-de Vries equation

Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Remarks

- Traditional spectral solvers require a smaller time-increment to achieve similar accuracy
- The authors chose q (hyperparameter) using a tolerance ϵ (in this case set to machine precision)

$$q = 0.5 \log \epsilon / \log(\Delta t)$$

Korteweg-de Vries equation

Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Remarks

- Traditional spectral solvers require a smaller time-increment to achieve similar accuracy
- The authors chose q (hyperparameter) using a tolerance ϵ (in this case set to machine precision)

$$q = 0.5 \log \epsilon / \log(\Delta t)$$

- Time step for this problem is $\Delta t = 0.6$

Korteweg-de Vries equation

Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Remarks

- Traditional spectral solvers require a smaller time-increment to achieve similar accuracy
- The authors chose q (hyperparameter) using a tolerance ϵ (in this case set to machine precision)

$$q = 0.5 \log \epsilon / \log(\Delta t)$$

- Time step for this problem is $\Delta t = 0.6$
- For the case of noise-free training data, the error in estimating λ_1 and λ_2 is 0.023%, and 0.006%, respectively, while the case with 1% noise in the training data returns errors of 0.057%, and 0.017%, respectively.

Korteweg-de Vries equation

Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Remarks

- Traditional spectral solvers require a smaller time-increment to achieve similar accuracy
- The authors chose q (hyperparameter) using a tolerance ϵ (in this case set to machine precision)

$$q = 0.5 \log \epsilon / \log(\Delta t)$$

- Time step for this problem is $\Delta t = 0.6$
- For the case of noise-free training data, the error in estimating λ_1 and λ_2 is 0.023%, and 0.006%, respectively, while the case with 1% noise in the training data returns errors of 0.057%, and 0.017%, respectively.
- Even for large temporal variations in the solution, the model is able to resolve the dynamics accurately

Table of Contents

Conclusion

Questions

Conclusion

- Introduced physics-informed neural networks, a new class of universal function approximators that are capable of encoding any underlying physical laws that govern a given data-set (described by PDEs)

Questions

Conclusion

- Introduced physics-informed neural networks, a new class of universal function approximators that are capable of encoding any underlying physical laws that govern a given data-set (described by PDEs)
- Design data-driven algorithms for inferring solutions to general nonlinear PDEs, and constructing computationally efficient physics-informed surrogate models.

Questions

Conclusion

- Introduced physics-informed neural networks, a new class of universal function approximators that are capable of encoding any underlying physical laws that govern a given data-set (described by PDEs)
- Design data-driven algorithms for inferring solutions to general nonlinear PDEs, and constructing computationally efficient physics-informed surrogate models.

Questions

- How deep/wide should the neural network be ? How much data is really needed ?

Conclusion

- Introduced physics-informed neural networks, a new class of universal function approximators that are capable of encoding any underlying physical laws that govern a given data-set (described by PDEs)
- Design data-driven algorithms for inferring solutions to general nonlinear PDEs, and constructing computationally efficient physics-informed surrogate models.

Questions

- How deep/wide should the neural network be ? How much data is really needed ?
- Why does the algorithm converge to unique values for the parameters of the differential operators, i.e., why is the algorithm not suffering from local optima for the parameters of the differential operator?

Conclusion

- Introduced physics-informed neural networks, a new class of universal function approximators that are capable of encoding any underlying physical laws that govern a given data-set (described by PDEs)
- Design data-driven algorithms for inferring solutions to general nonlinear PDEs, and constructing computationally efficient physics-informed surrogate models.

Questions

- How deep/wide should the neural network be ? How much data is really needed ?
- Why does the algorithm converge to unique values for the parameters of the differential operators, i.e., why is the algorithm not suffering from local optima for the parameters of the differential operator?
- Does the network suffer from vanishing gradients for deeper architectures and higher order differential operators? Could this be mitigated by using different activation functions?

Conclusion

- Introduced physics-informed neural networks, a new class of universal function approximators that are capable of encoding any underlying physical laws that govern a given data-set (described by PDEs)
- Design data-driven algorithms for inferring solutions to general nonlinear PDEs, and constructing computationally efficient physics-informed surrogate models.

Questions

- How deep/wide should the neural network be ? How much data is really needed ?
- Why does the algorithm converge to unique values for the parameters of the differential operators, i.e., why is the algorithm not suffering from local optima for the parameters of the differential operator?
- Does the network suffer from vanishing gradients for deeper architectures and higher order differential operators? Could this be mitigated by using different activation functions?
- Can we improve on initializing the network weights or normalizing the data? Loss function choices (MSE, SSE)? Robustness?

Code

Code

- https://github.com/maziarraissi/PINNs (TensorFlow)

Code

- https://github.com/maziarraissi/PINNs (TensorFlow)
- Blog: https://maziarraissi.github.io/PINNs/

Code

- https://github.com/maziarraissi/PINNs (TensorFlow)
- Blog: https://maziarraissi.github.io/PINNs/
- NeuralPDE.jl: https://neuralpde.sciml.ai/dev/

Code

- https://github.com/maziarraissi/PINNs (TensorFlow)
- Blog: https://maziarraissi.github.io/PINNs/
- NeuralPDE.jl: https://neuralpde.sciml.ai/dev/
- IDRLNet: https://github.com/idrl-lab/idrlnet (PyTorch)

References

Raissi, M., Perdikaris, P., and Karniadakis, G. E.
Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017a.
URL https://arxiv.org/abs/1711.10561.

Raissi, M., Perdikaris, P., and Karniadakis, G. E.
Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, 2017b.
URL https://arxiv.org/abs/1711.10566.

Raissi, M., Perdikaris, P., and Karniadakis, G. E.
Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.
Journal of Computational Physics, 378:686–707, 2019.
URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.