DL in Applied Mathematics

Lecture 14:

Physics Informed Neural Network

*

Content

What is PINN?

Physics-informed neural networks (PINNs) are a type of universal function approximators that can embed the knowledge of any physical laws that govern a given data-set in the learning process, and can be described by partial differential equations (PDEs). They overcome the low data availability of some biological and engineering systems that makes most state-of-the-art machine learning techniques lack robustness, rendering them ineffective in these scenarios.

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

- Data-driven solution
- Data-driven discovery of PDEs

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

History

Physics-informed machine learning debuted in the 1990s, appearing in scattered papers throughout the decade. A resurgence in machine learning around 2010 breathed new life into this promising offshoot.

Another milestone came in 2014 with Facebook’s DeepFace. Its algorithm was able to detect whether two faces in unfamiliar photos are of the same person with 97.25% accuracy, despite differing lighting conditions or angles. Humans generally have an average of 97.53% accuracy, meaning Facebook's facial-processing software had nearly the same accuracy as a human being.

The Physics-Informed Neural Networks (PINN) approach was first introduced by Raissi et al. in their 2019 paper "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations". The paper was published in the Journal of Computational Physics and presented a novel framework for solving partial differential equations (PDEs) using neural networks.

Raissi et al. proposed a method for incorporating known physical laws, represented by the PDE, into the neural network architecture. This allowed the network to learn both from the data and from the underlying physics of the system being modeled, leading to more accurate and physically meaningful predictions.

Since its introduction, the PINN approach has gained widespread attention in the scientific community and has been applied to a wide range of problems in various fields such as fluid mechanics, heat transfer, and materials science, among others.

Strengths and limitations of physics-informed machine learning

One of the greatest strengths in physics-informed machine learning is that it yields results quickly—in just a fraction of a second. Because of the flow structure of the neural network, the output of a new sample is very efficient. And when the training is successful, the predictions are impressively accurate.

A neural network can be successfully trained to recognize a collection of drugs and make accurate predictions about its effectiveness. But this same system will be far less effective when it comes to predictions about drug compounds, because it cannot successfully predict anything about drugs with which it is unfamiliar.

There are several advantages of using Physics-Informed Neural Networks (PINN) compared to traditional methods of solving physical problems

* Incorporation of physics-based constraints: PINN allows for the incorporation of physical principles or equations as constraints during the neural network training process. This helps to ensure that the predictions made by the neural network are physically meaningful and accurate.
* Ability to handle noisy and incomplete data: Traditional methods of solving physical problems often struggle with noisy or incomplete data. PINN can handle such data by learning the underlying physical relationships and using them to make accurate predictions.
* Faster computation: PINN can often solve physical problems much faster than traditional methods. This is because the neural network can be trained on large datasets and can make predictions quickly once it has been trained.
* Versatility: PINN can be applied to a wide range of physical problems, from fluid dynamics to materials science to quantum mechanics. This makes it a versatile tool for researchers working in various fields of physics.
* Reduced need for manual intervention: Traditional methods of solving physical problems often require manual intervention, such as setting up boundary conditions or adjusting parameters. PINN can automate many of these tasks, reducing the need for human intervention and speeding up the modeling process.

Some disadvantages of PINN

* Limited interpretability: The complexity of neural networks can make it difficult to understand how the model arrived at its predictions. This can be a disadvantage in applications where interpretability is important, such as in scientific research where physical principles need to be understood and validated.
* Data requirements: PINN requires a large amount of data to be effective, especially in cases where the physical problem is highly complex. Obtaining this data can be time-consuming and expensive, and may not always be feasible.
* Computationally intensive: PINN can be computationally intensive, especially when dealing with large datasets or complex physical problems. This can lead to longer training times, higher energy consumption, and the need for more powerful computing resources.
* Sensitivity to hyperparameters: The performance of PINN is highly dependent on the choice of hyperparameters, such as the number of layers, the number of neurons in each layer, and the learning rate. Selecting the right hyperparameters can be challenging and time-consuming.
* Difficulty in implementation: Implementing PINN requires expertise in both neural networks and physics, which can be a barrier for researchers who are not familiar with both fields. Additionally, PINN may require custom code or modifications to existing neural network frameworks, which can add to the complexity of implementation.

The formula for PINN

* The neural network architecture: This includes the number of layers, the number of neurons in each layer, the activation function used in each neuron, and any other parameters related to the neural network structure.

Loss function

In Physics-Informed Neural Networks (PINN), the mean squared error (MSE) is a commonly used loss function for the data-driven component of the loss function. The MSE measures the average squared difference between the predicted output of the neural network and the actual output, and is given by the formula:

$MSE = 1/N * \Sigma i=1 \text{ to } N (yi - \hat{y}i)^2$

where N is the number of training samples, yi is the actual output for sample i, and ŷi is the predicted output for sample i.

In PINN, the MSE is typically combined with a physics-driven loss function that incorporates physical principles or equations. This allows the neural network to learn not only from the data, but also from the underlying physical laws governing the system being modeled.

Optimizer

* The optimizer: The optimizer is used to adjust the weights and biases of the neural network during training in order to minimize the loss function. Popular optimizers include stochastic gradient descent (SGD), Adam, and RMSprop.

The physics-informed part

- The physics-informed part: This is the part of the formula that incorporates physical principles or equations into the neural network training process. This can involve adding constraints or boundary conditions to the loss function, or incorporating physical laws as part of the neural network structure.

The general aim

- The general aim of this class(subject) is to set the foundations for a new paradigm in modeling and computation that enriches deep learning with the longstanding developments in mathematical physics. These developments are presented in the context of two main problem classes: data-driven solution and data-driven discovery of partial differential equations. To this end, let us consider parametrized and nonlinear partial differential equations of the general form
- $u_t + N[u; \lambda] = 0$,
- where $u(t, x)$ denotes the latent (hidden) solution and $N[\cdot; \lambda]$ is a nonlinear operator parametrized by λ. This setup encapsulates a wide range of problems in mathematical physics including conservation laws, diffusion processes, advection-diffusion-reaction systems, and kinetic equations.

Example

- Let’s take as a example the one dimensional Burgers’ equation

As an example, let us consider the Burgers' equation. In one space dimension, the Burger's equation along with Dirichlet boundary conditions reads as

[illegible]

[illegible]

[illegible]

Let us define [illegible] to be given by

[illegible]

and proceed by approximating [illegible]u(t,x) by a deep neural network. To highlight the simplicity in implementing this idea let us include a Python code snippet using Tensorflow. To this end, [illegible] can be simply defined as

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

Correspondingly, the physics informed neural network $f(t, x)$ takes the form

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

The shared parameters between the neural networks $u(t,x)$ and $f(t,x)$ can be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_f,$$

where

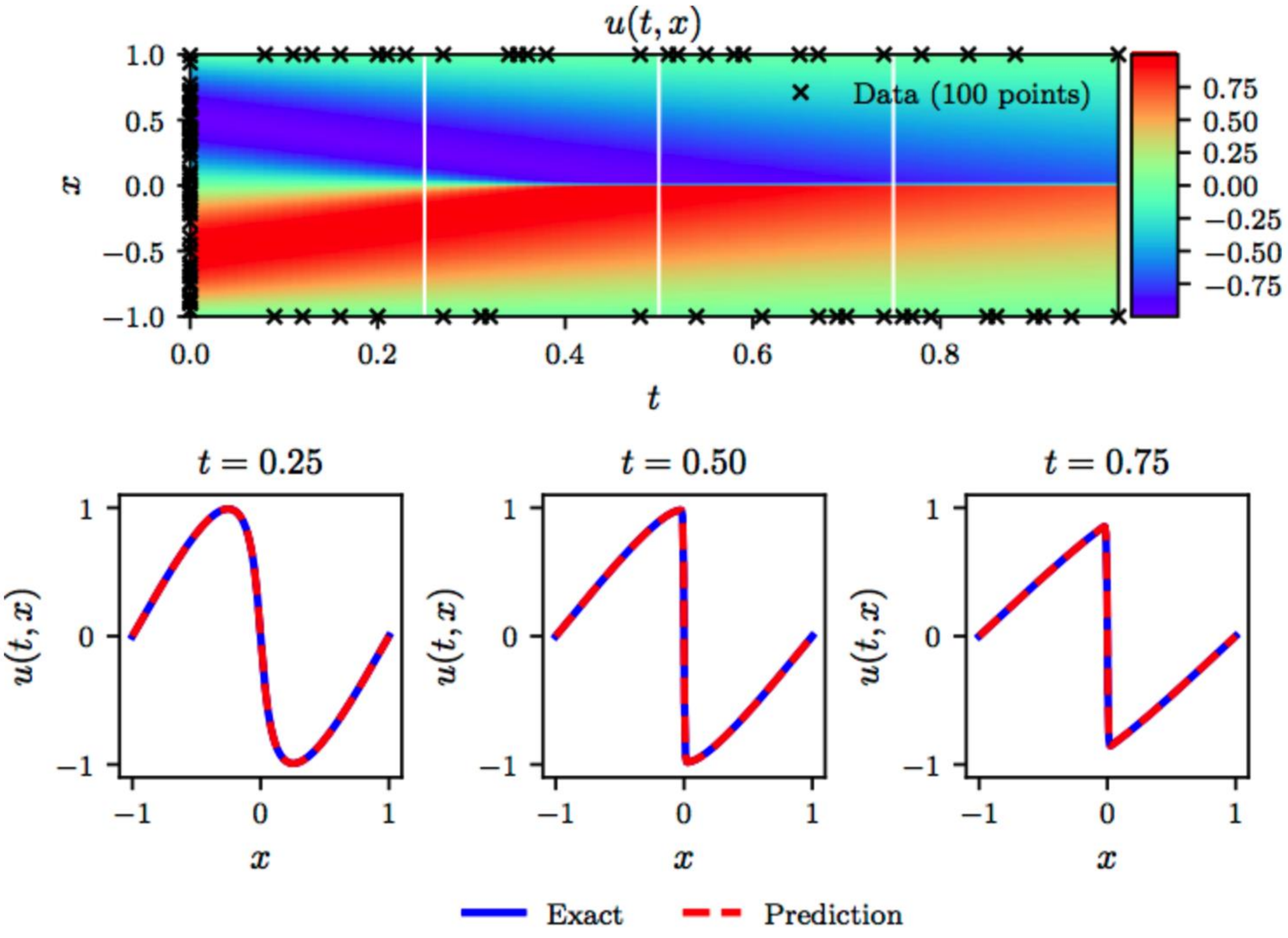
$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

and

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

Here, $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(t,x)$ and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ specify the collocations points for $f(t,x)$. The loss MSE_u corresponds to the initial and boundary data while MSE_f enforces the structure imposed by the Burgers' equation at a finite set of collocation points.

The following figure summarizes our results for the data-driven solution of the Burgers' equation.



Example (Shrödinger 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:

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

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

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`)
- $\{x_0^i, h_0^i\}_{i=1}^{N_0}$ 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

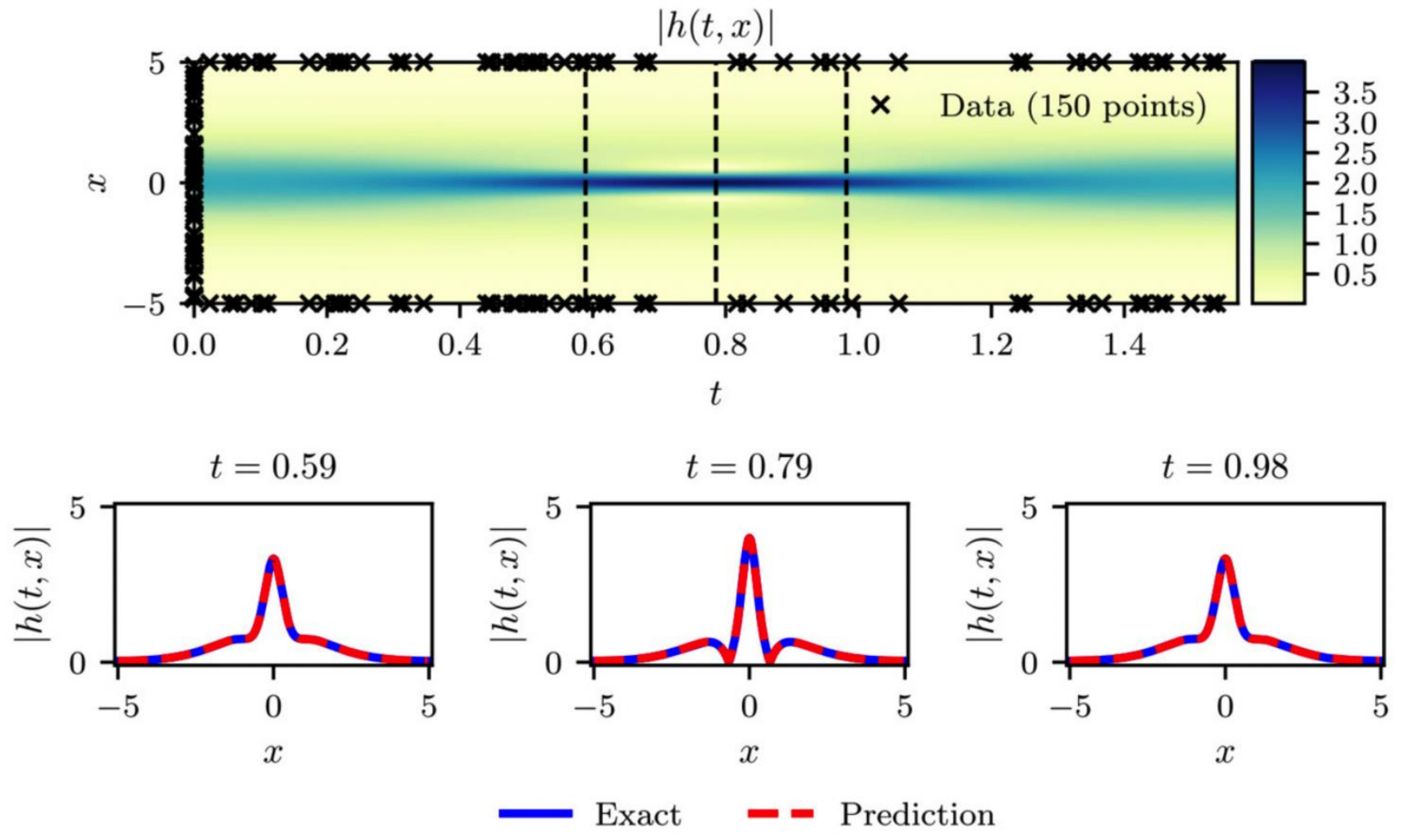


Figure: Top: Boundary and Initial data (150 points), Bottom: Snapshots of the solution of the Schrödinger equation using a PINN

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

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

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

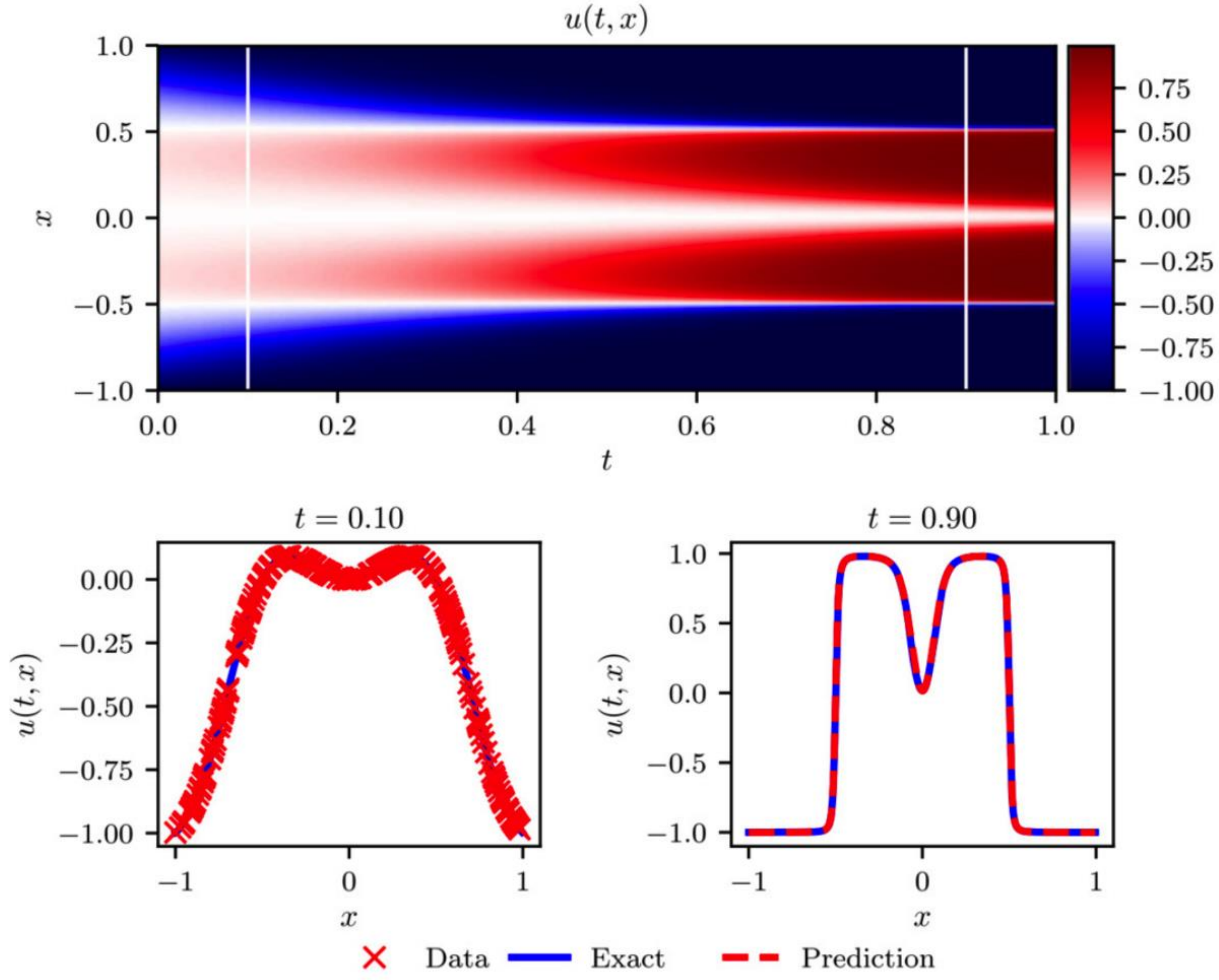
- Make use of the above adaptive time-stepper

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

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

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

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


$u(t,x)$
x
t
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$t = 0.10$
$t = 0.90$
$u(t,x)$
0.00
−0.25
−0.50
−0.75
−1.00
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
x
Data
Exact
Prediction

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: $\left\{t^i, x^i, y^i, u^i, v^i\right\}_{i=1}^{N}$

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

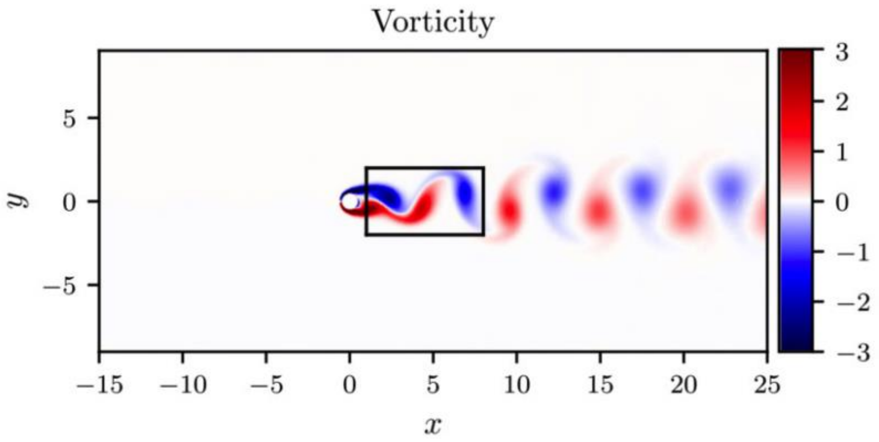
- Learn $\lambda = \{\lambda_1, \lambda_2\}$, and pressure field $p(t,x,y)$ by jointly approximating $[\psi(t,x,y) \quad p(t,x,y)]$ with a single NN with two outputs

Navier-Stokes Equations

- 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 $p(t,x,y)$



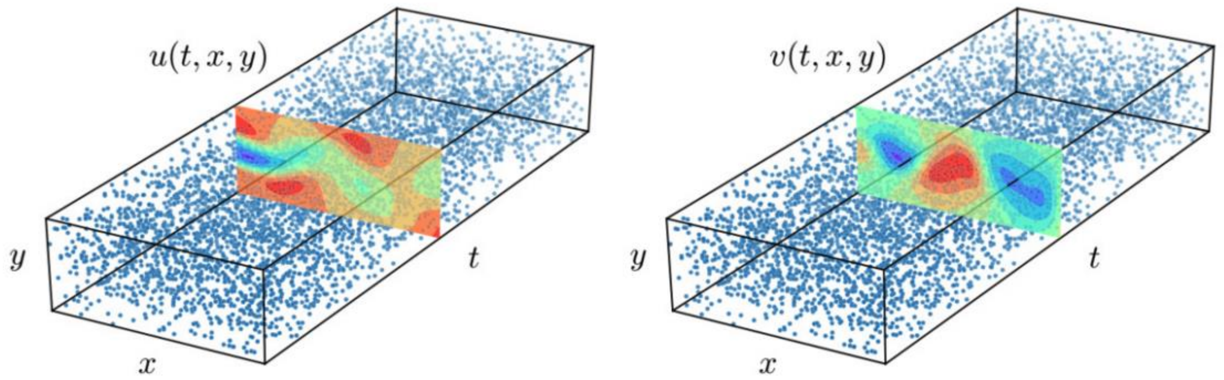


Figure: Navier-Stokes equation: **Top**: Incompressible flow and dynamic vortex shedding past a circular cylinder at Re $= 100$. The spatio-temporal training data correspond to the depicted rectangular region in the cylinder wake. **Bottom**: Locations of training data-points for the stream-wise and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

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

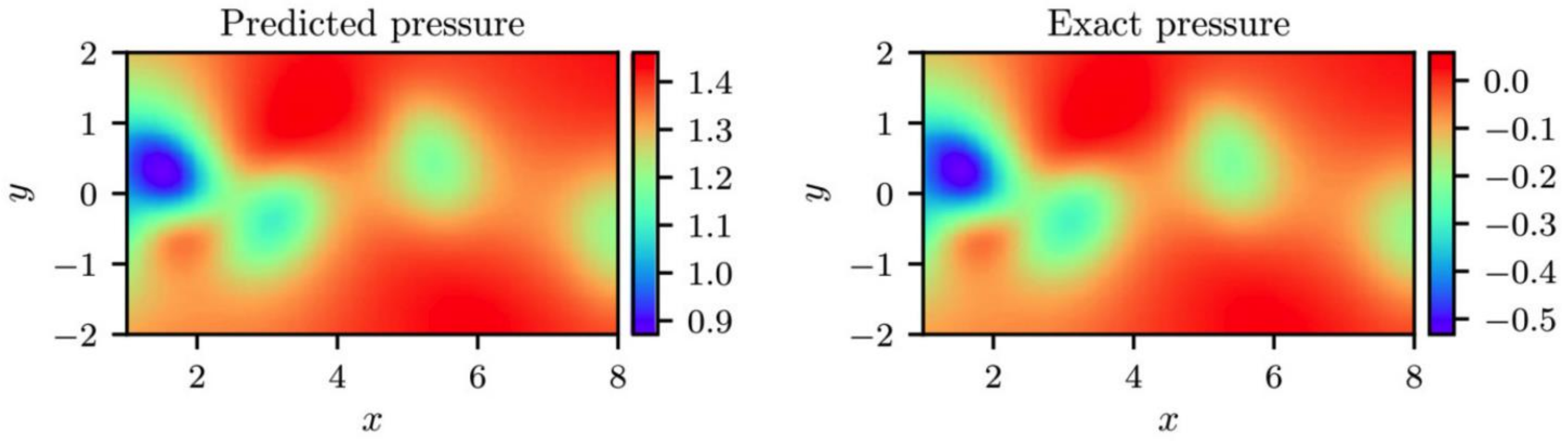


Figure: Results for predicted pressure field

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 $p(t, x, y)$.

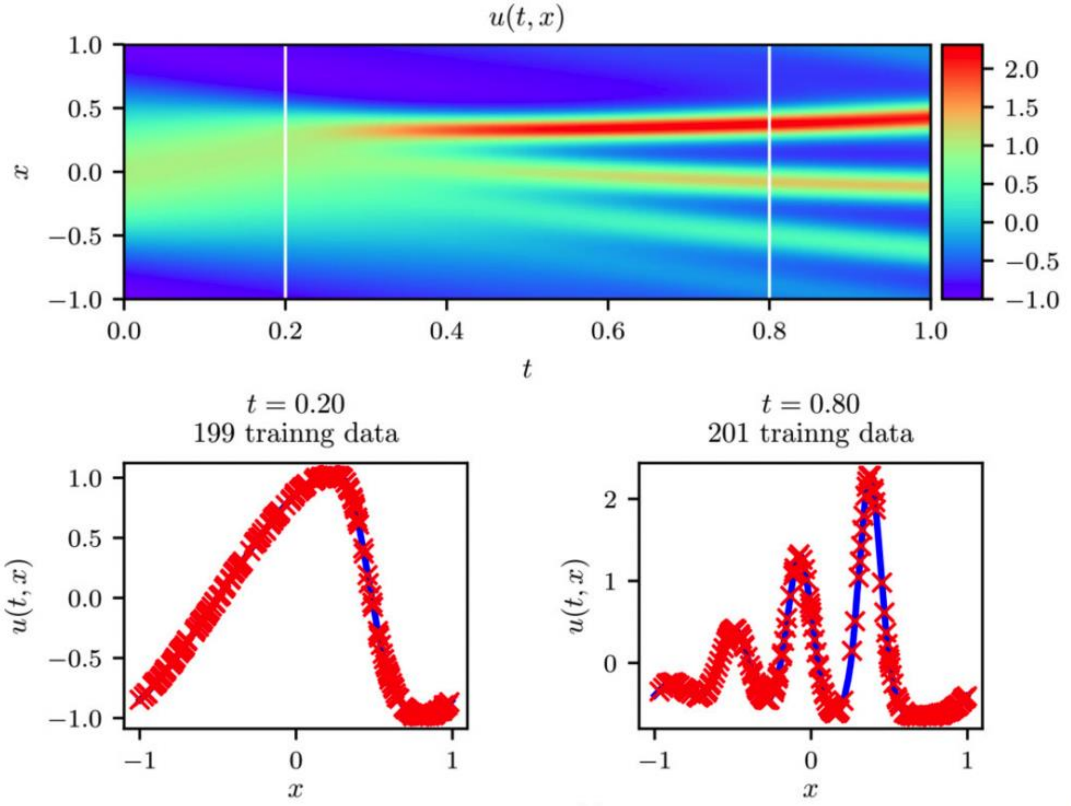
Example (Korteweg–de Vries Equation)

- 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)

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



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

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.