AI in the Sciences and Engineering

PINNs – Limitations and Extensions – Part 1

Spring Semester 2024

Siddhartha Mishra
Ben Moseley
ETH zürich

Recap - what is a PINN?

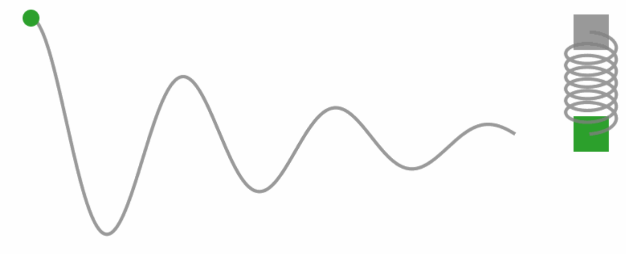
Damped harmonic oscillator:

$$m\frac{d^2u}{dt^2}+\mu\frac{du}{dt}+ku=0$$

Initial conditions:

$$u(t=0)=1$$
$$u_t(t=0)=0$$

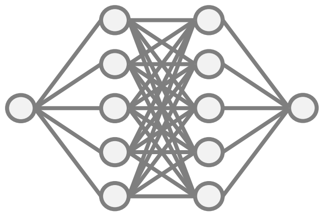
$u=$ displacement
$m=$ mass of oscillator
$\mu=$ coefficient of friction
$k=$ spring constant



Key idea: use a neural network to directly approximate the solution

$$NN(t;\theta)\approx u(t)$$

t



$$NN(t;\theta)\approx u(t)$$

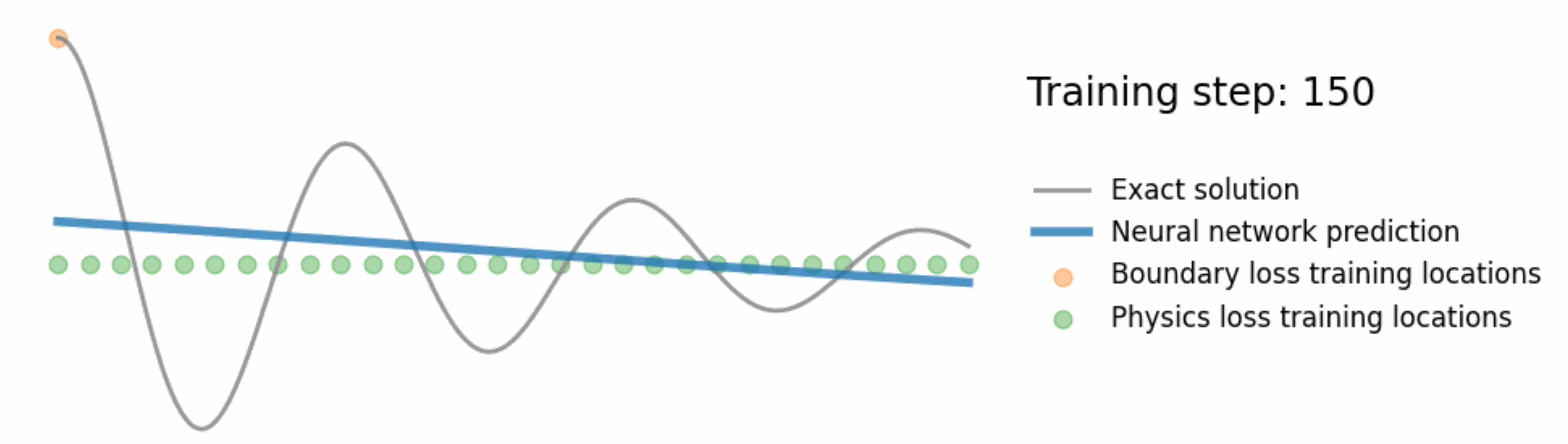
Train the network using the loss function:

Boundary loss $L_b(\theta)$

$$L(\theta)=\lambda_1(NN(t=0;\theta)-1)^2+\lambda_2\left(\frac{dNN}{dt}(t=0;\theta)-0\right)^2$$

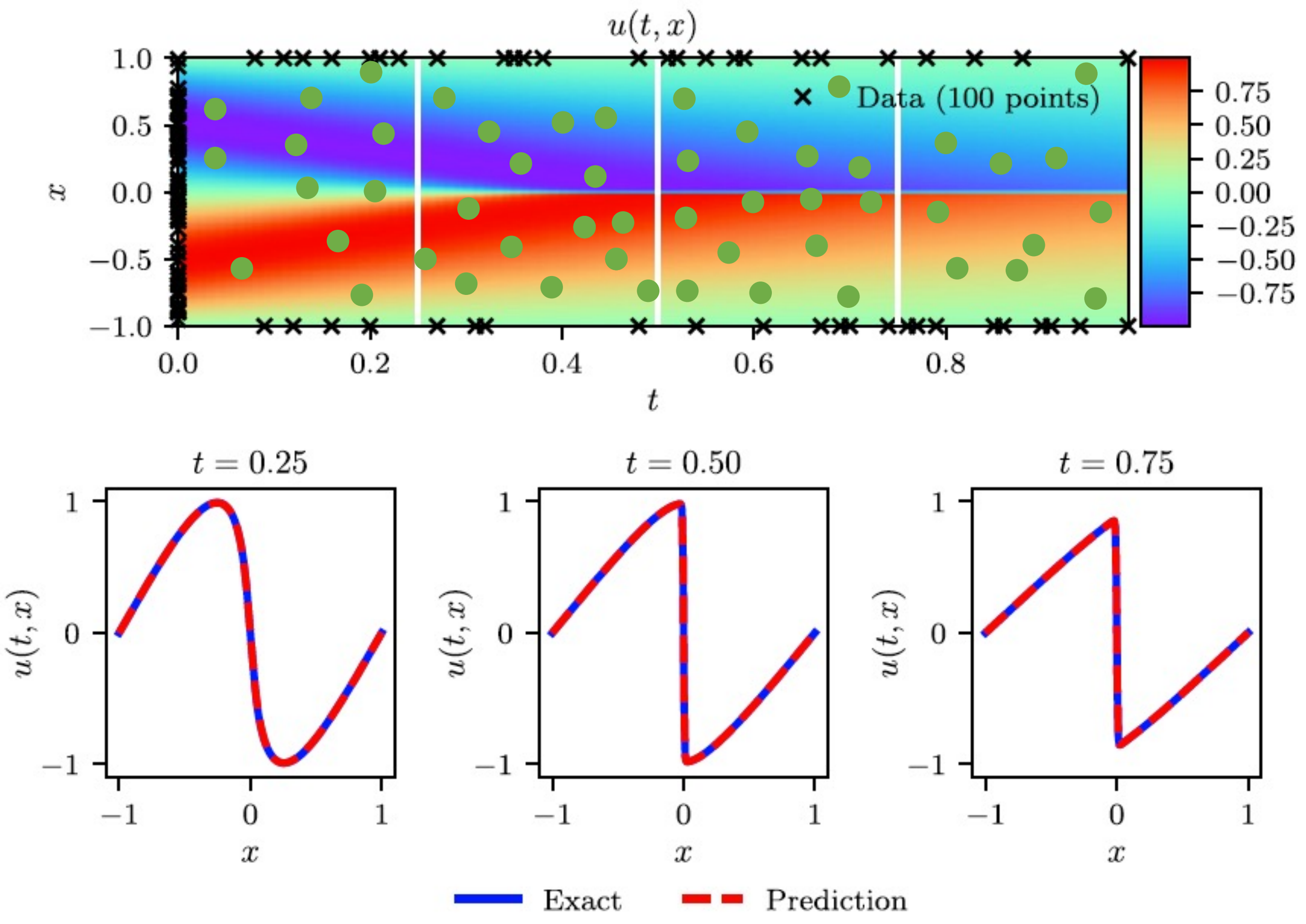
Physics loss $L_p(\theta)$ (aka PDE residual)

$$+\frac{1}{N_p}\sum_i^{N_p}\left(\left[m\frac{d^2}{dt^2}+\mu\frac{d}{dt}+k\right]NN(t_i;\theta)\right)^2$$



Raissi et al, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP (2018)
Lagaris et al, Artificial neural networks for solving ordinary and partial differential equations, IEEE (1998)

Recap – PINNs for solving Burgers' equation



Raissi et al, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP (2018)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$

$$u(x,0) = -\sin(\pi x)$$
$$u(-1,t) = u(+1,t) = 0$$

$$L(\theta) = L_b(\theta) + L_p(\theta)$$

$$L_b(\theta) = \frac{\lambda_1}{N_{b1}}\sum_j^{N_{b1}}\left(NN\left(\underline{x_j},0;\theta\right) + \underline{\sin(\pi x_j)}\right)^2 + \frac{\lambda_2}{N_{b2}}\sum_k^{N_{b2}}\left(NN(-1,\underline{t_k};\theta) - \underline{0}\right)^2 + \frac{\lambda_3}{N_{b3}}\sum_l^{N_{b3}}\left(NN(+1,\underline{t_l};\theta) - \underline{0}\right)^2$$

$$L_p(\theta) = \frac{1}{N_p}\sum_i^{N_p}\left(\left(\frac{\partial NN}{\partial t} + NN\frac{\partial NN}{\partial x} - \nu\frac{\partial^2 NN}{\partial x^2}\right)(\underline{x_i,t_i};\theta)\right)^2$$

Recap – PINN applications

PINNs for solving **forward** simulation:

$$L(\theta) = L_b(\theta) + L_p(\theta)$$

$$L_b(\theta) = \sum_k \frac{\lambda_k}{N_{bk}} \sum_j^{N_{bk}} \left\| \mathcal{B}_k\left[NN(x_{kj};\theta)\right] - g_k(x_{kj}) \right\|^2$$

$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \| \mathcal{D}[NN(x_i;\theta)] - f(x_i) \|^2$$

PINNs for solving **inverse** problems:

$$L(\theta, \phi) = L_p(\theta, \phi) + L_d(\theta)$$

$$L_p(\theta, \phi) = \frac{1}{N_p} \sum_i^{N_p} \| \mathcal{D}[NN(x_i;\theta);\phi] - f(x_i) \|^2$$

$$L_d(\theta) = \frac{\lambda}{N_d} \sum_l^{N_d} \| NN(x_l;\theta) - u_l \|^2$$

PINNs for **equation discovery**:

$$L(\theta, \Lambda) = L_p(\theta, \Lambda) + L_d(\theta)$$

$$L_p(\theta, \Lambda) = \frac{1}{N_p} \sum_i^{N_p} \| \Lambda \phi[NN(x_i;\theta)] \|^2 + \| \Lambda \|^2$$

$$L_d(\theta) = \frac{\lambda}{N_d} \sum_l^{N_d} \| NN(x_l;\theta) - u_l \|^2$$

Course timeline

Tutorials		Lectures			
Mon 12:15-14:00 HG E 5		*Wed 08:15-10:00 ML H 44*		*Fri 12:15-13:00 ML H 44*	
19.02.		21.02.	~~Course introduction~~	23.02.	~~Introduction to deep learning I~~
26.02.	~~Introduction to PyTorch~~	28.02.	~~Introduction to deep learning II~~	01.03.	~~Importance of PDEs in science~~
04.03.	~~CNNs and surrogate modelling~~	06.03.	~~Introduction to physics-informed neural networks~~	08.03.	**PINNs – limitations and extensions**
11.03.	Implementing PINNs I	13.03.	PINNs – extensions and theory	15.03.	PINNs – theory
18.03.	Implementing PINNs II	20.03.	Introduction to operator learning	22.03.	DeepONets and spectral neural operators
25.03.	Operator learning I	27.03.	Fourier- and convolutional- neural operators	29.03.	
01.04.		03.04.		05.04.	
08.04.	Operator learning II	10.04.	Operator learning – limitations and extensions	12.04.	Introduction to transformers
15.04.		17.04.	Foundational models for operator learning	19.04.	Graph neural networks for PDEs
22.04.	GNNs	24.04.	GNNs for PDEs / introduction to diffusion models	26.04.	Introduction to diffusion models
29.04.	Transformers	01.05.		03.05.	Diffusion models - applications
06.05.	Diffusion models	08.05.	Introduction to differentiable physics	10.05.	Hybrid workflows
13.05.	Coding autodiff from scratch	15.05.	Neural differential equations	17.05.	Introduction to JAX
20.05.		22.05.	Symbolic regression and equation discovery	24.05.	Course summary and future trends
27.05.	Introduction to JAX / NDEs	29.05.	Guest lecture: ML in chemistry and biology	31.05.	Guest lecture: ML in chemistry and biology

Overview of lectures

- PINN limitations
- PINN extensions for improving:
  - Computational cost
  - Convergence / accuracy
  - Scalability to more complex problems
- Summary: when should I use PINNs?

Part 1

Part 2

Overview of lectures

- PINN limitations
- PINN extensions for improving:
  - Computational cost
  - Convergence / accuracy
  - Scalability to more complex problems
- Summary: when should I use PINNs?

Part 1

Part 2

Learning objectives

- Explain the advantages and disadvantages of PINNs
- Understand current research directions on improving their performance

Advantages / limitations of PINNs

Advantages

- **Mesh-free**
- Can jointly solve **forward** and **inverse** problems
- Often performs well on “messy” problems (where some observational data is available)
- Tractable, analytical solution gradients (e.g. for sensitivity analysis)
- Mostly **unsupervised**

Limitations

- **?**

PINN limitation 1) – computational cost

PINNs for solving wave equation

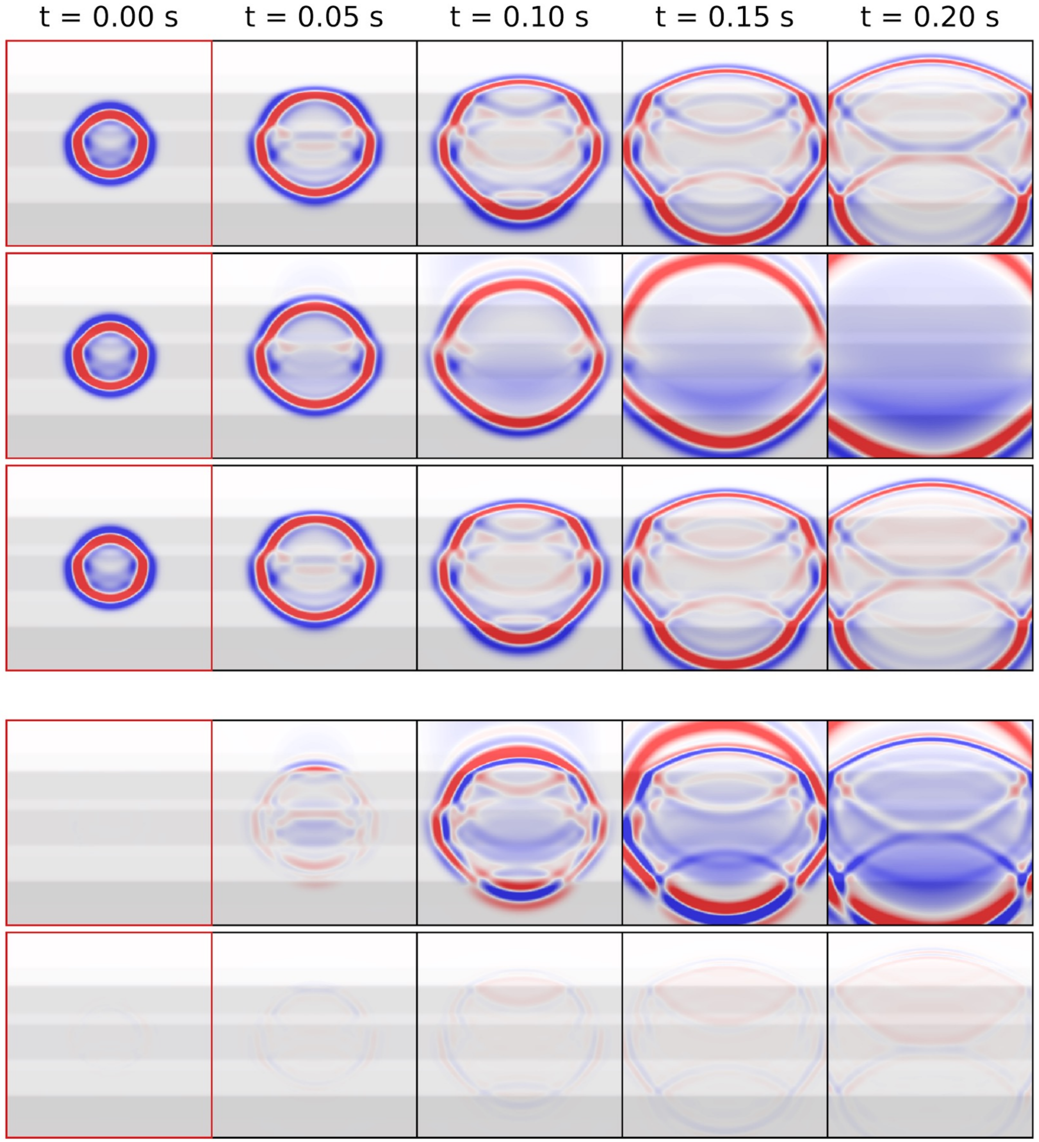
Ground truth FD simulation

"Naïve" NN

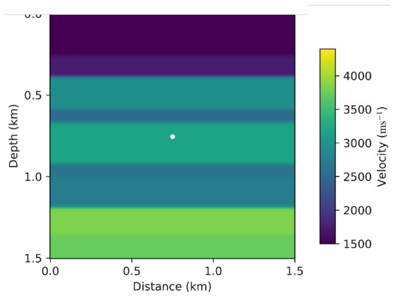
PINN

Difference (NN)

Difference (PINN)



Velocity model, $c(x)$



Moseley et al, Solving the wave equation with physics-informed deep learning, ArXiv (2020)

Mini-batch size $N_b = N_p = 500$ (random sampling)
Fully connected network with 10 layers, 1024 hidden units
Softplus activation
Adam optimiser

Training time: **~1 hour**

PINNs for solving wave equation

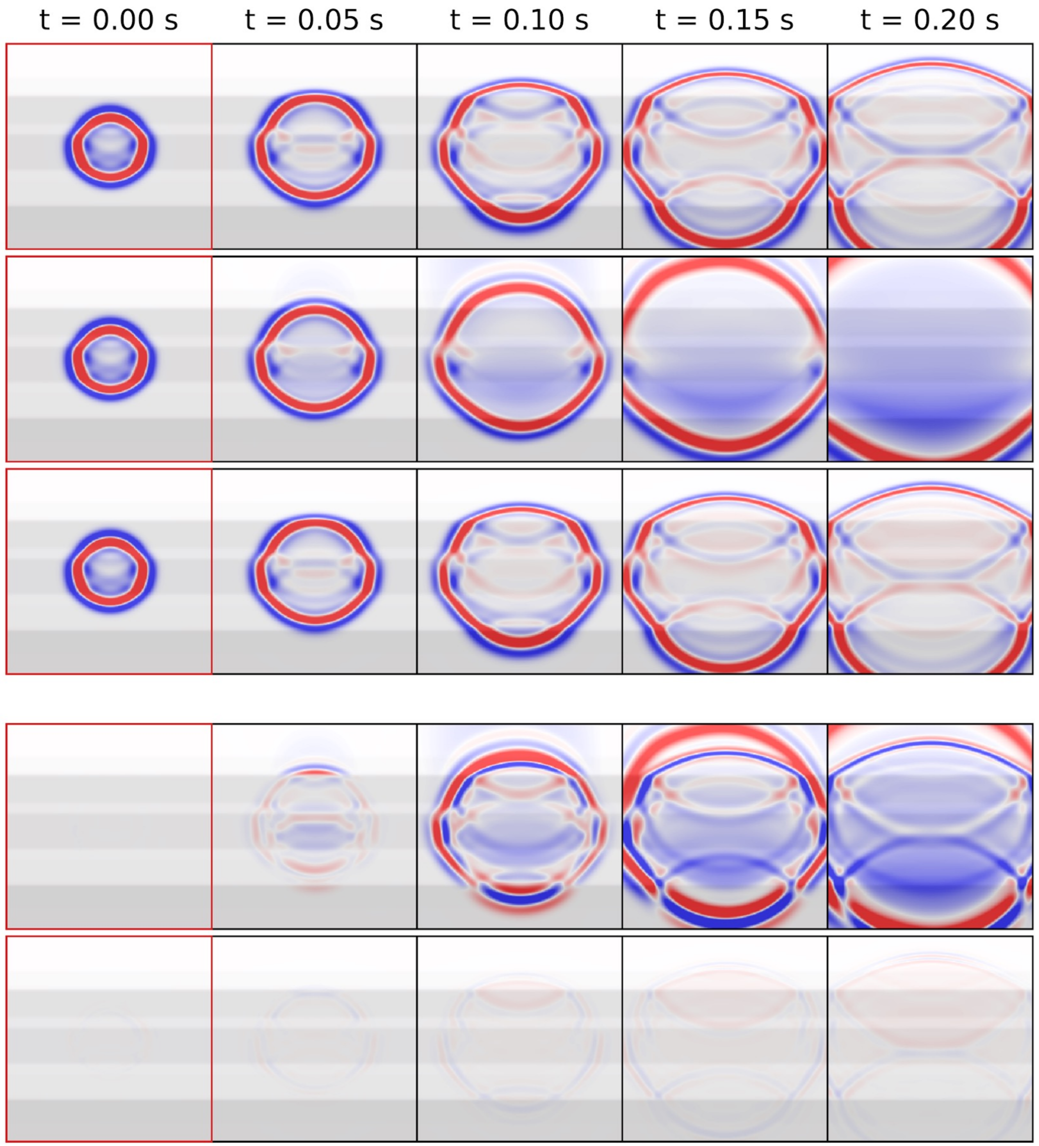
Ground truth FD simulation

"Naïve" NN

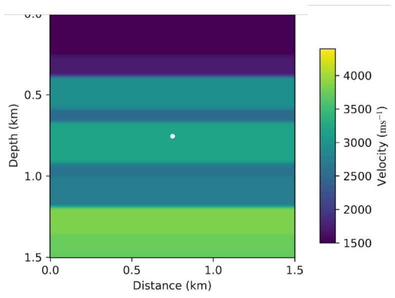
PINN

Difference (NN)

Difference (PINN)



Velocity model, $c(x)$



Moseley et al, Solving the wave equation with physics-informed deep learning, ArXiv (2020)

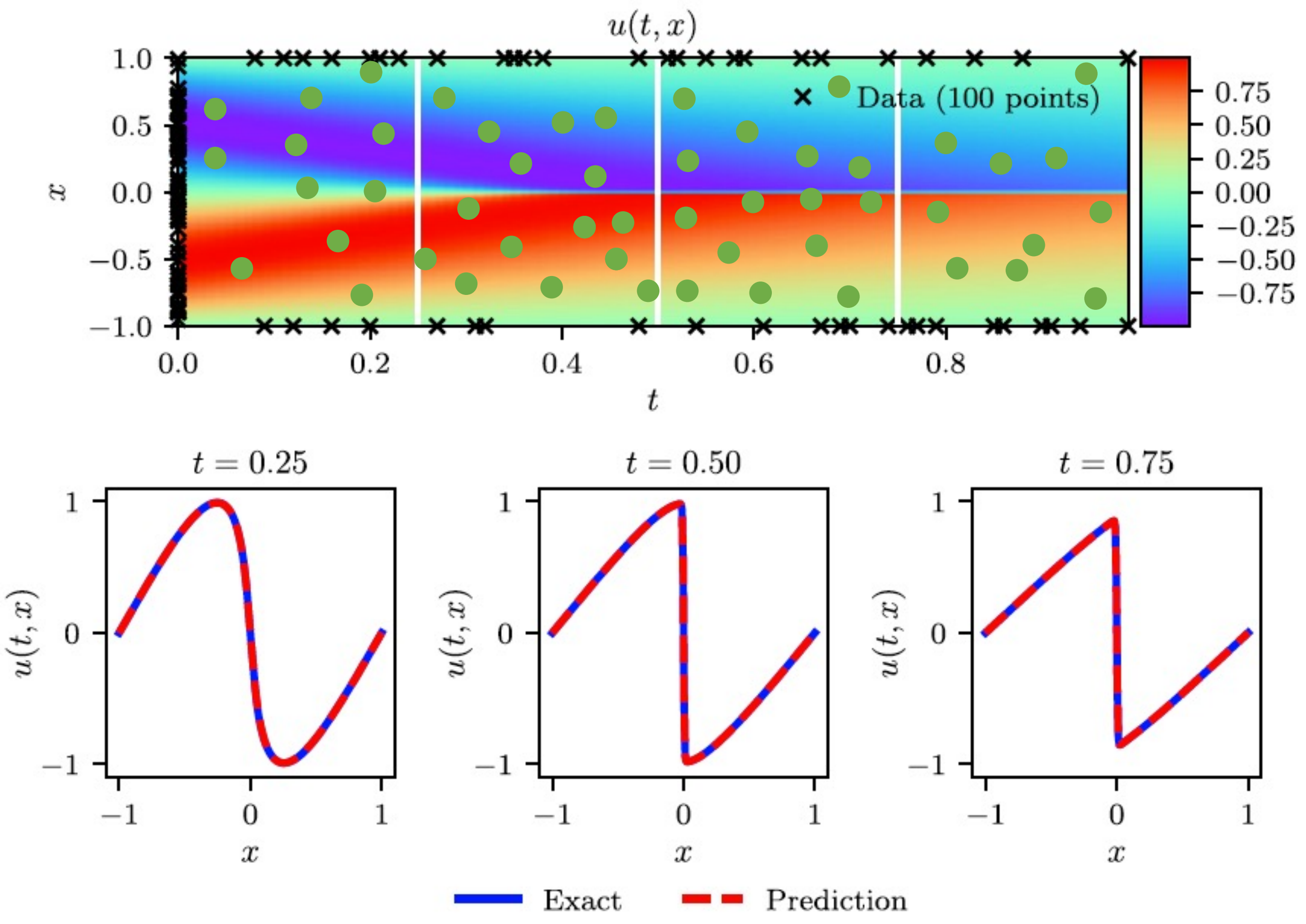
Mini-batch size $N_b = N_p = 500$ (random sampling)
Fully connected network with 10 layers, 1024 hidden units
Softplus activation
Adam optimiser

Training time: **~1 hour**

PINNs need to be **retrained** for each new I/BC !

PINN limitation 2) – poor convergence

Competing loss terms

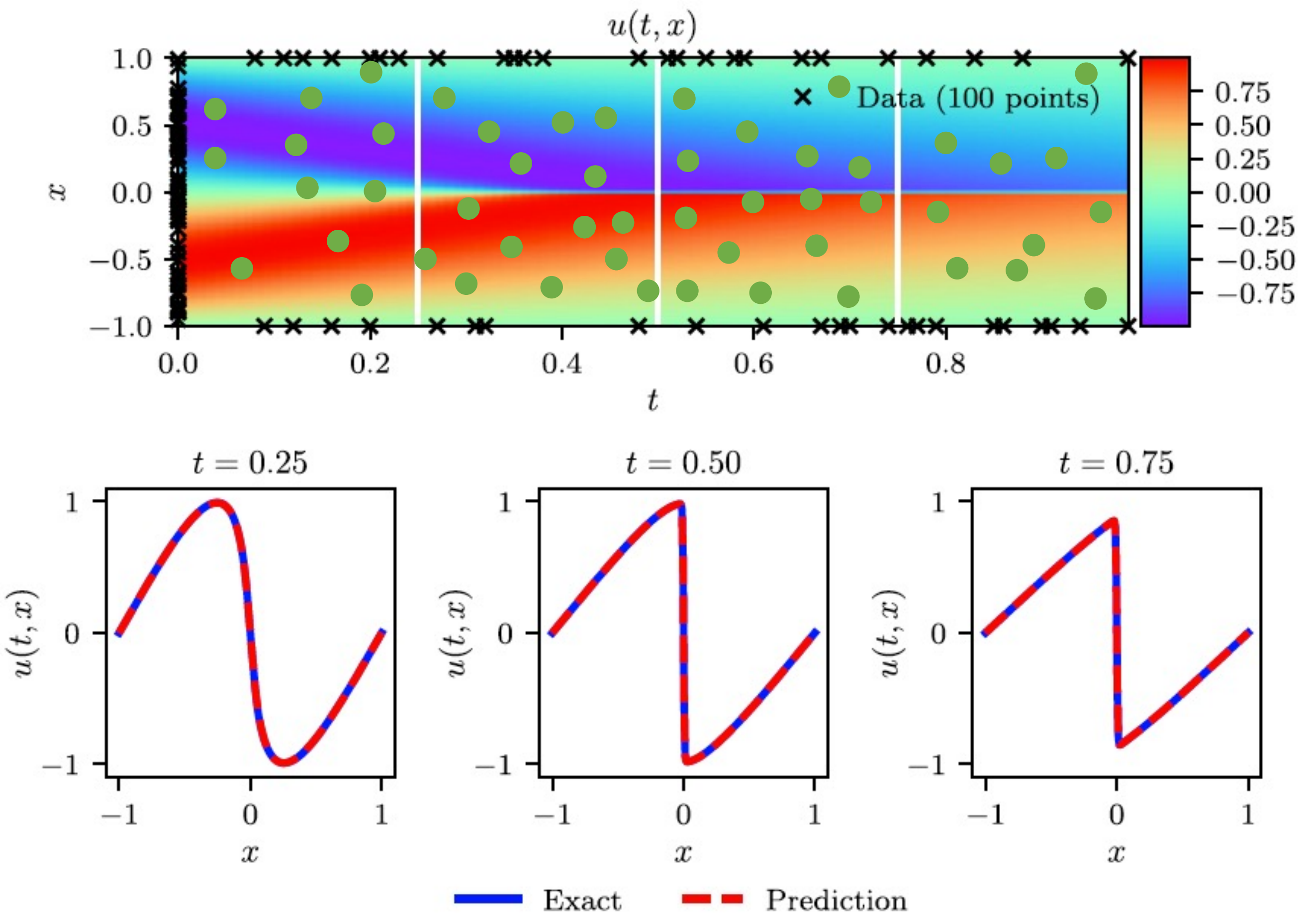


$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_{j}^{N_{b1}} \left(NN(\underline{x_j}, 0; \theta) + \underline{\sin(\pi x_j)}\right)^2$$
$$+ \frac{\lambda_2}{N_{b2}} \sum_{k}^{N_{b2}} \left(NN(-1, \underline{t_k}; \theta) - \underline{0}\right)^2$$
$$+ \frac{\lambda_3}{N_{b3}} \sum_{l}^{N_{b3}} \left(NN(+1, \underline{t_l}; \theta) - \underline{0}\right)^2$$

$$L_p(\theta) = \frac{1}{N_p} \sum_{i}^{N_p} \left(\left(\frac{\partial NN}{\partial t} + NN \frac{\partial NN}{\partial x} - \nu \frac{\partial^2 NN}{\partial x^2} \right) (\underline{x_i, t_i}; \theta) \right)^2$$

Raissi et al, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP (2018)

Competing loss terms



Raissi et al, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP (2018)

$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_j^{N_{b1}} \left(NN(\underline{x_j}, 0; \theta) + \underline{\sin(\pi x_j)}\right)^2 + \frac{\lambda_2}{N_{b2}} \sum_k^{N_{b2}} \left(NN(-1, \underline{t_k}; \theta) - \underline{0}\right)^2 + \frac{\lambda_3}{N_{b3}} \sum_l^{N_{b3}} \left(NN(+1, \underline{t_l}; \theta) - \underline{0}\right)^2$$

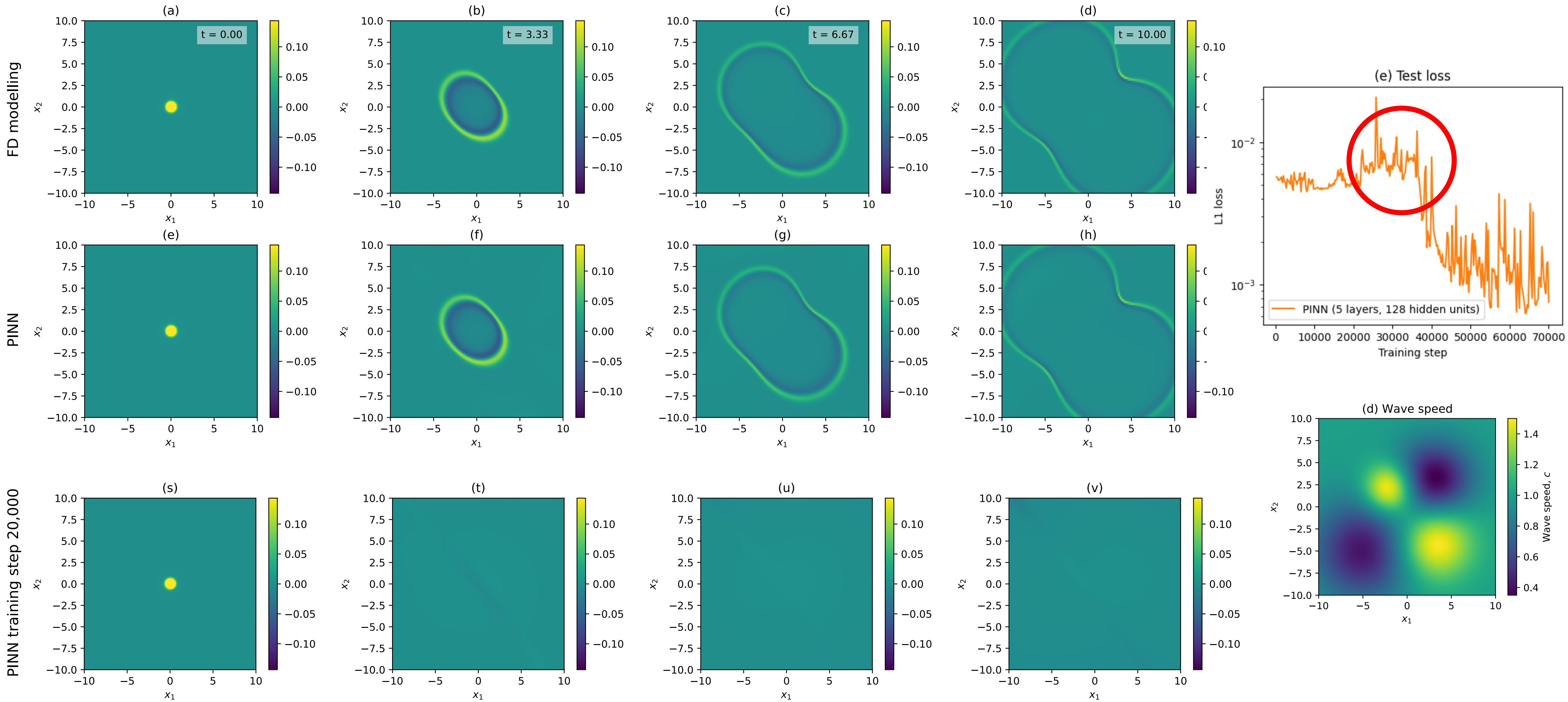
$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \left(\left(\frac{\partial NN}{\partial t} + NN \frac{\partial NN}{\partial x} - \nu \frac{\partial^2 NN}{\partial x^2} \right) (\underline{x_i, t_i}; \theta) \right)^2$$

How do we choose λ_1, λ_2, and λ_3?

λ too small => doesn't learn unique solution
λ too large => only learns boundary condition

Thus, there can be **competing** terms in the loss function

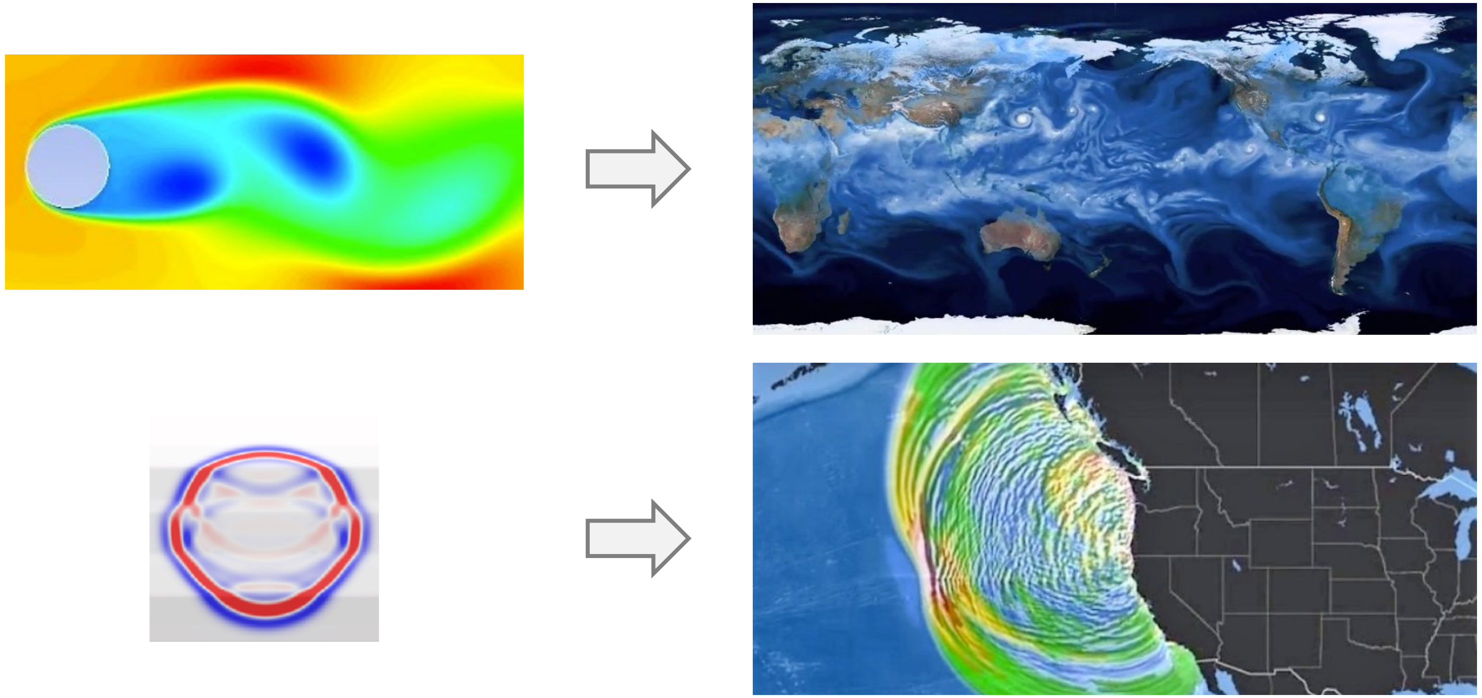
PINN solution “thrashing”



Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ACM (2023)

PINN limitation 3) – scaling to more complex problems

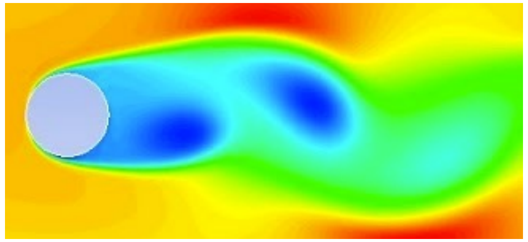
Scaling to more complex problems

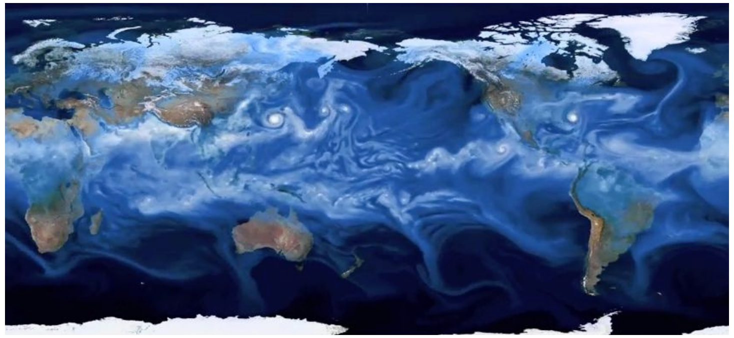


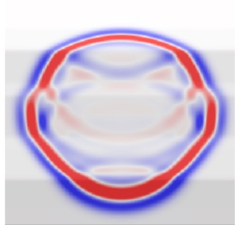
Majority of PINN research focuses on **toy/simplified** problems,
as proof-of-principle studies

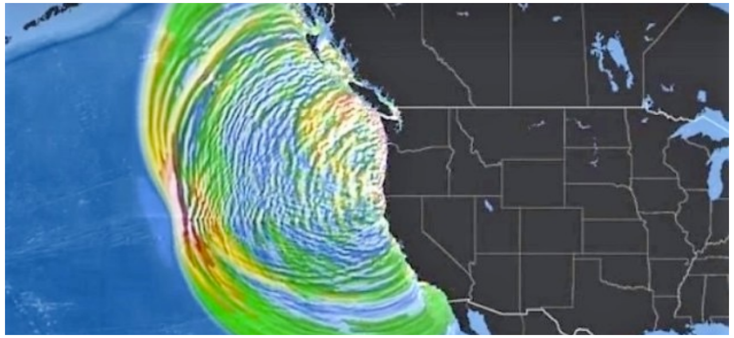
Image credits: Lawrence Berkeley National Laboratory / NOAA / NWS / Pacific Tsunami Warning Center

Scaling to more complex problems









Majority of PINN research focuses on **toy/simplified** problems,
as proof-of-principle studies

It is often challenging to **scale** traditional scientific algorithms to:

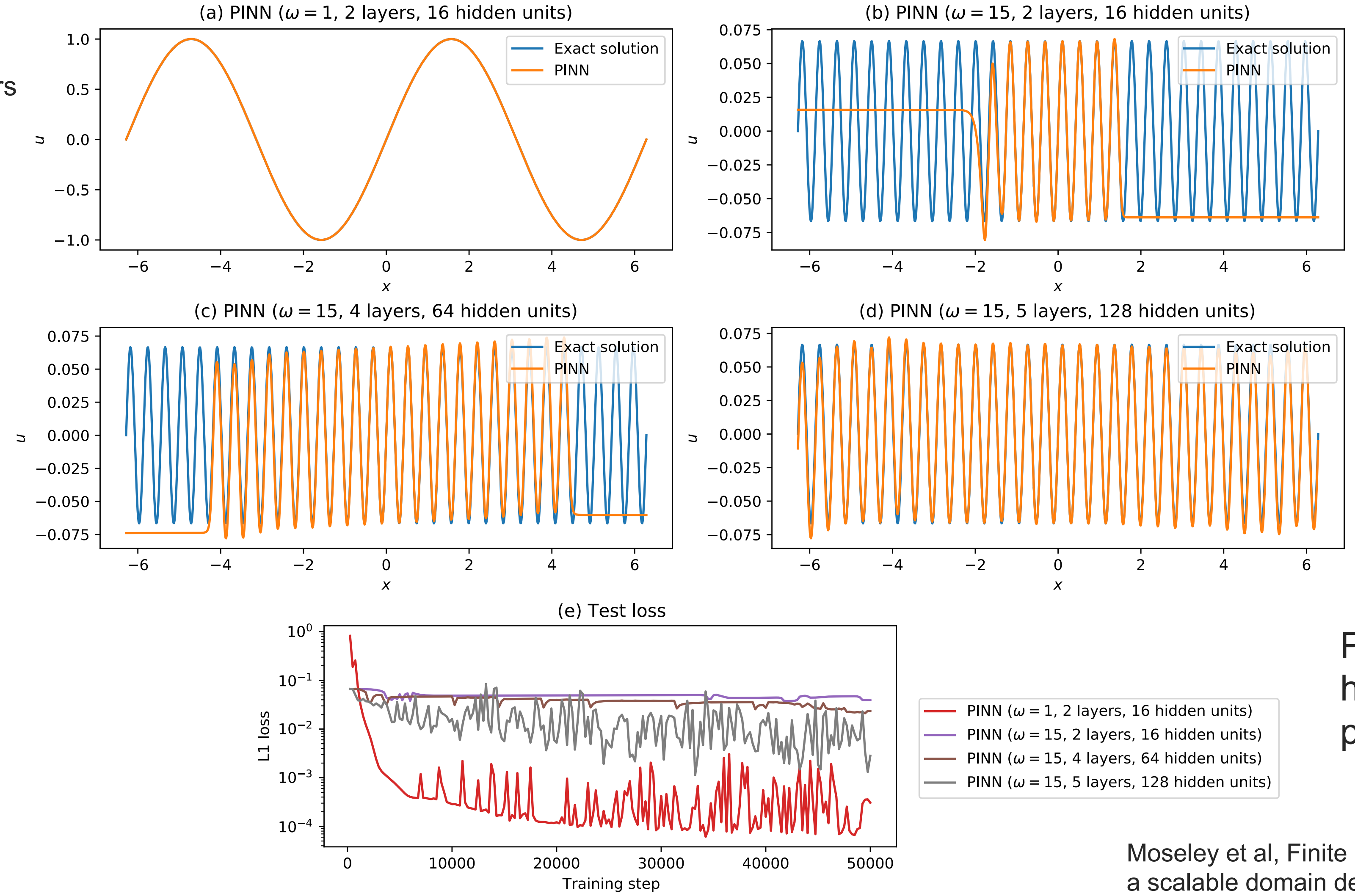
- More complex phenomena (**multi-scale**, **multi-physics**)
- Large domains / higher frequency solutions
- Incorporate **real, noisy** and **sparse** data

How do PINNs cope in this setting?

Image credits: Lawrence Berkeley National Laboratory / NOAA / NWS / Pacific Tsunami Warning Center

Scaling PINNs to higher frequencies

321 free parameters



PINN solving:

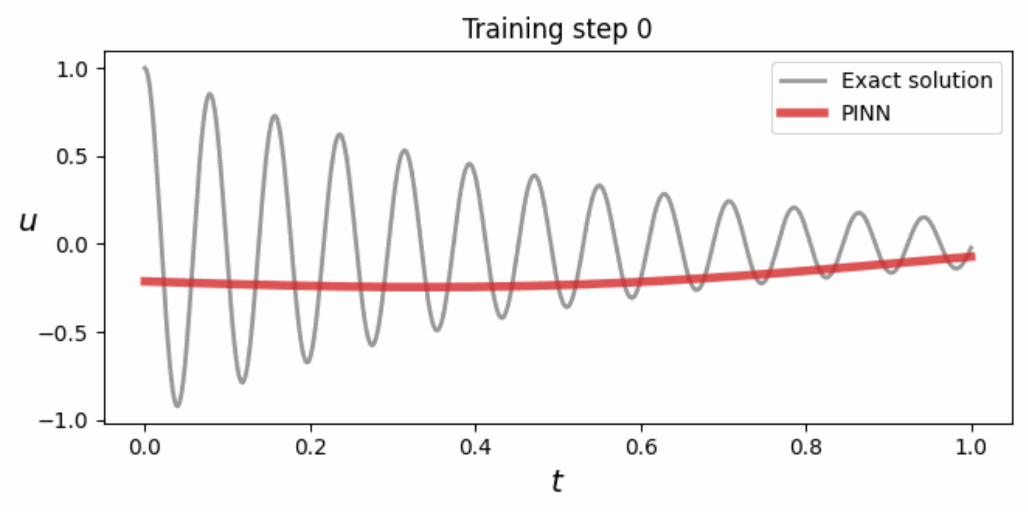
$$\frac{du}{dx} = \cos(\omega x)$$
$$u(0) = 0$$

66,433 free parameters

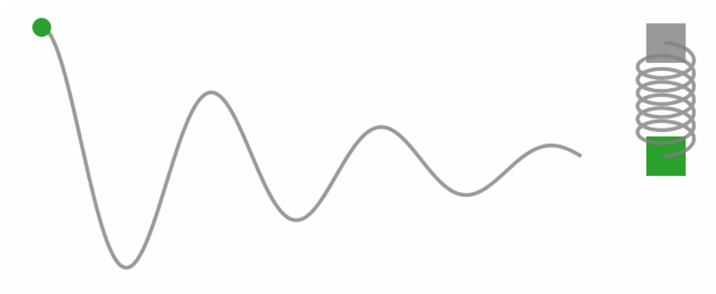
Problem: PINNs **struggle** to solve high-frequency / multiscale problems

Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ACM (2023)

Scaling PINNs to higher frequencies



Network size: 2 hidden layers, 64 hidden units



Damped harmonic oscillator

Problem: PINNs **struggle** to solve high-frequency / multiscale problems

Advantages / limitations of PINNs

Advantages

- **Mesh-free**
- Can jointly solve **forward** and **inverse** problems
- Often performs well on “messy” problems (where some observational data is available)
- Tractable, analytical solution gradients (e.g. for sensitivity analysis)
- Mostly **unsupervised**

Limitations

- **Computational cost** often high (especially for forward-only problems)
- Can be hard to **optimise** (and convergence properties less well understood)
- Challenging to **scale** to larger domains, multi-scale, multi-physics problems

But.. many PINN extensions exist!

PINNs – an entire research field

SPRINGER LINK

Find a journal Publish with us Track your research Search

Home > Journal of Scientific Computing > Article

Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next

Open access | Published: 26 July 2022

Volume 92, article number 88, (2022) Cite this article

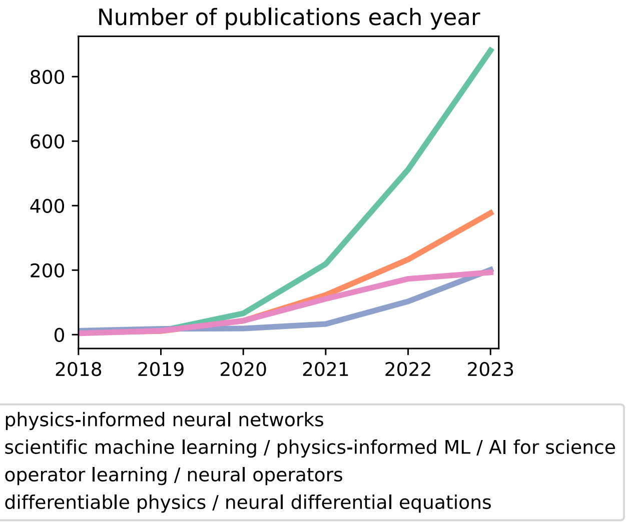
Download PDF

You have full access to this open access article

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi & Francesco Piccialli

Abstract

Physics-Informed Neural Networks (PINN) are neural networks (NNs) that encode model equations, like Partial Differential Equations (PDE), as a component of the neural network itself. PINNs are nowadays used to solve PDEs, fractional equations, integral-differential equations, and stochastic PDEs. This novel methodology has arisen as a multi-task learning framework in which a NN must fit observed data while reducing a PDE residual. This article provides a comprehensive review of the literature on PINNs: while the primary goal of the study was to characterize these networks and their related advantages and



Source: Scopus keyword search (Feb 2024)

[HTML] **Physics-informed neural networks**: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M Raissi, P Perdikaris, GE Karniadakis - Journal of Computational physics, 2019 - Elsevier

… We introduce **physics-informed neural networks** – **neural networks** that are trained to solve supervised learning tasks while respecting any given laws of physics described by general …

Save Cite Cited by 8017 Related articles All 7 versions

100-1000s of PINN applications / extensions!

Overview of lectures

- PINN limitations
- PINN extensions for improving:
  - Computational cost
  - Convergence / accuracy
  - Scalability to more complex problems
- Summary: when should I use PINNs?

Part 1

Part 2

Learning objectives

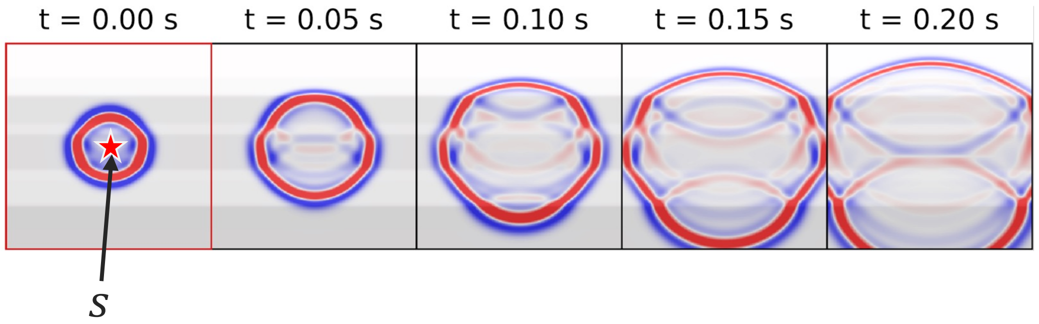
- Explain the advantages and disadvantages of PINNs
- Understand current research directions on improving their performance

Limitation 1) – computational cost

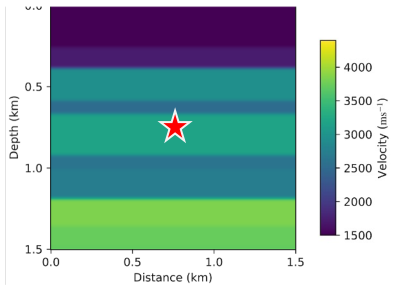
Conditioned PINNs

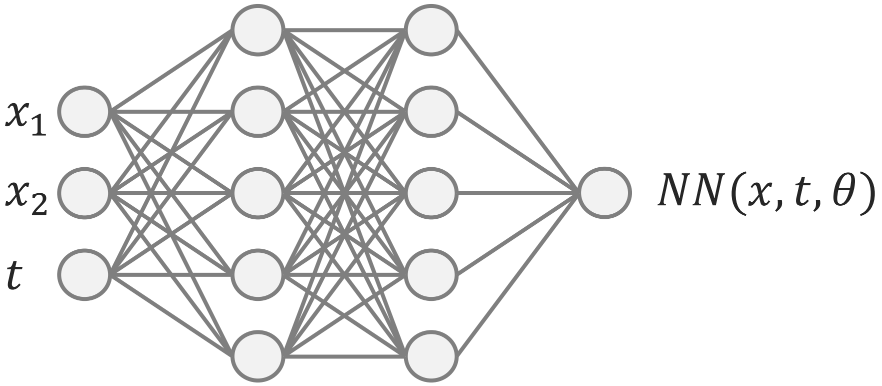
Idea: add I/BCs / other PDE parameters as **additional** network input parameters

Ground truth FD simulation



Velocity model, $c(x)$



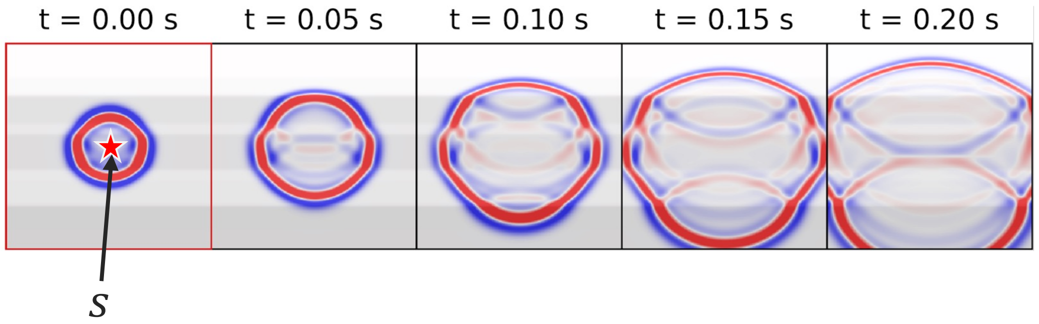


PINN

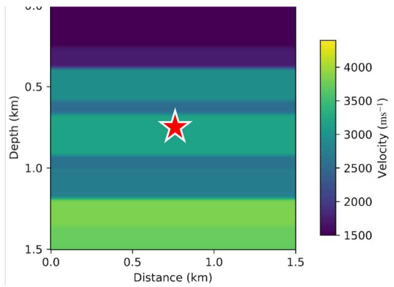
Conditioned PINNs

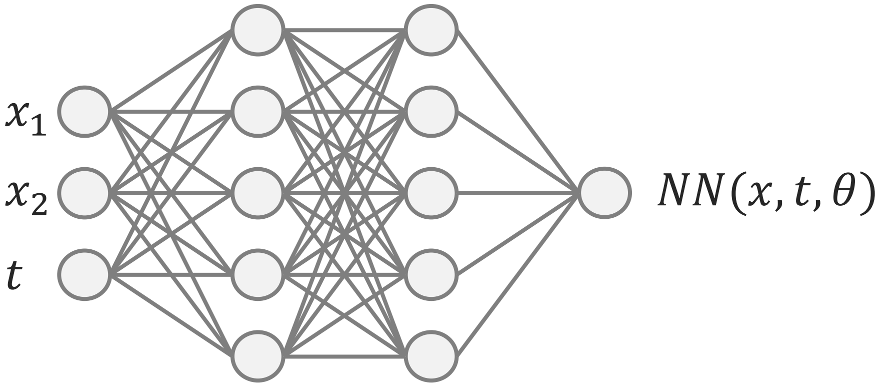
Idea: add I/BCs / other PDE parameters as **additional** network input parameters

Ground truth FD simulation

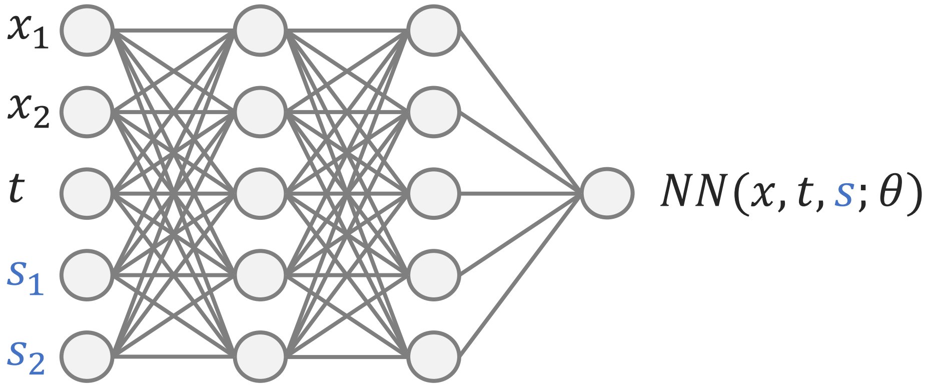


Velocity model, $c(x)$





PINN

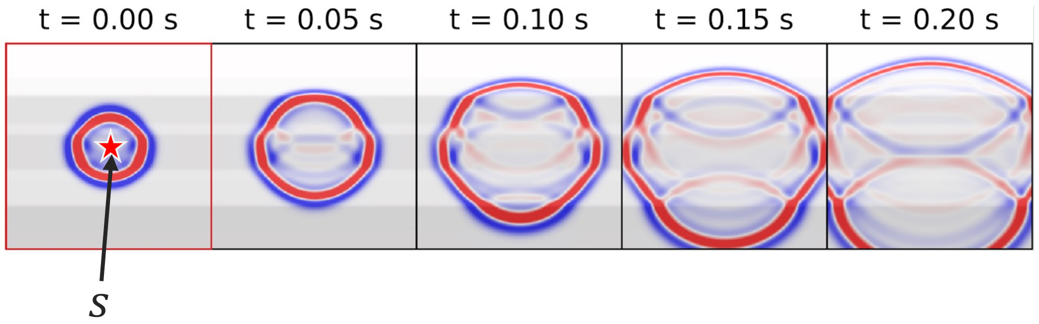


Conditioned PINN

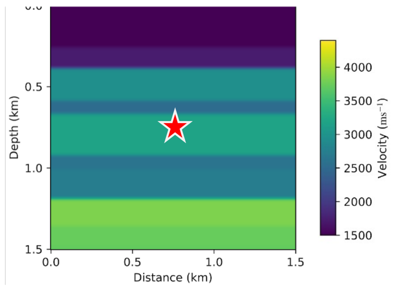
Conditioned PINNs

Idea: add I/BCs / other PDE parameters as **additional** network input parameters

Ground truth FD simulation



Velocity model, $c(x)$



$$L_b(\theta) = \frac{\lambda}{N_b} \sum_{j}^{N_b} \Big(NN(x_j, t_j; \theta) - u_{FD}(x_j, t_j) \Big)^2$$

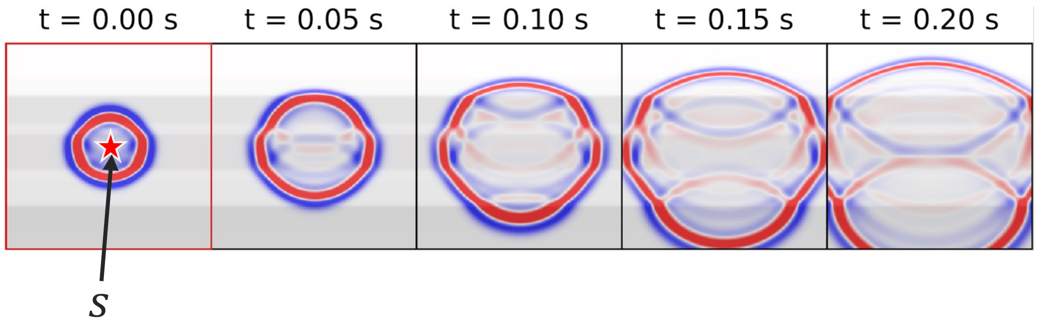
$$L_p(\theta) = \frac{1}{N_p} \sum_{i}^{N_p} \left(\left[\nabla^2 - \frac{1}{c(x_i)^2} \frac{\partial^2}{\partial t^2} \right] NN(x_i, t_i; \theta) \right)^2$$

PINN

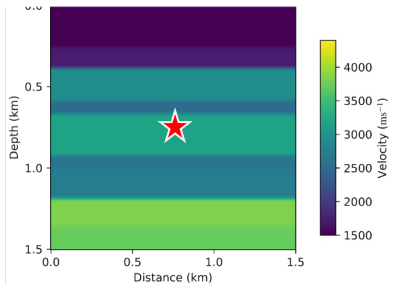
Conditioned PINNs

Idea: add I/BCs / other PDE parameters as **additional** network input parameters

Ground truth FD simulation



Velocity model, $c(x)$



Boundary examples from many **different** source locations

$$L_b(\theta) = \frac{\lambda}{N_b} \sum_{j}^{N_b} \Big(NN(x_j, t_j; \theta) - u_{FD}(x_j, t_j) \Big)^2$$

$$L_p(\theta) = \frac{1}{N_p} \sum_{i}^{N_p} \left(\left[\nabla^2 - \frac{1}{c(x_i)^2} \frac{\partial^2}{\partial t^2} \right] NN(x_i, t_i; \theta) \right)^2$$

PINN

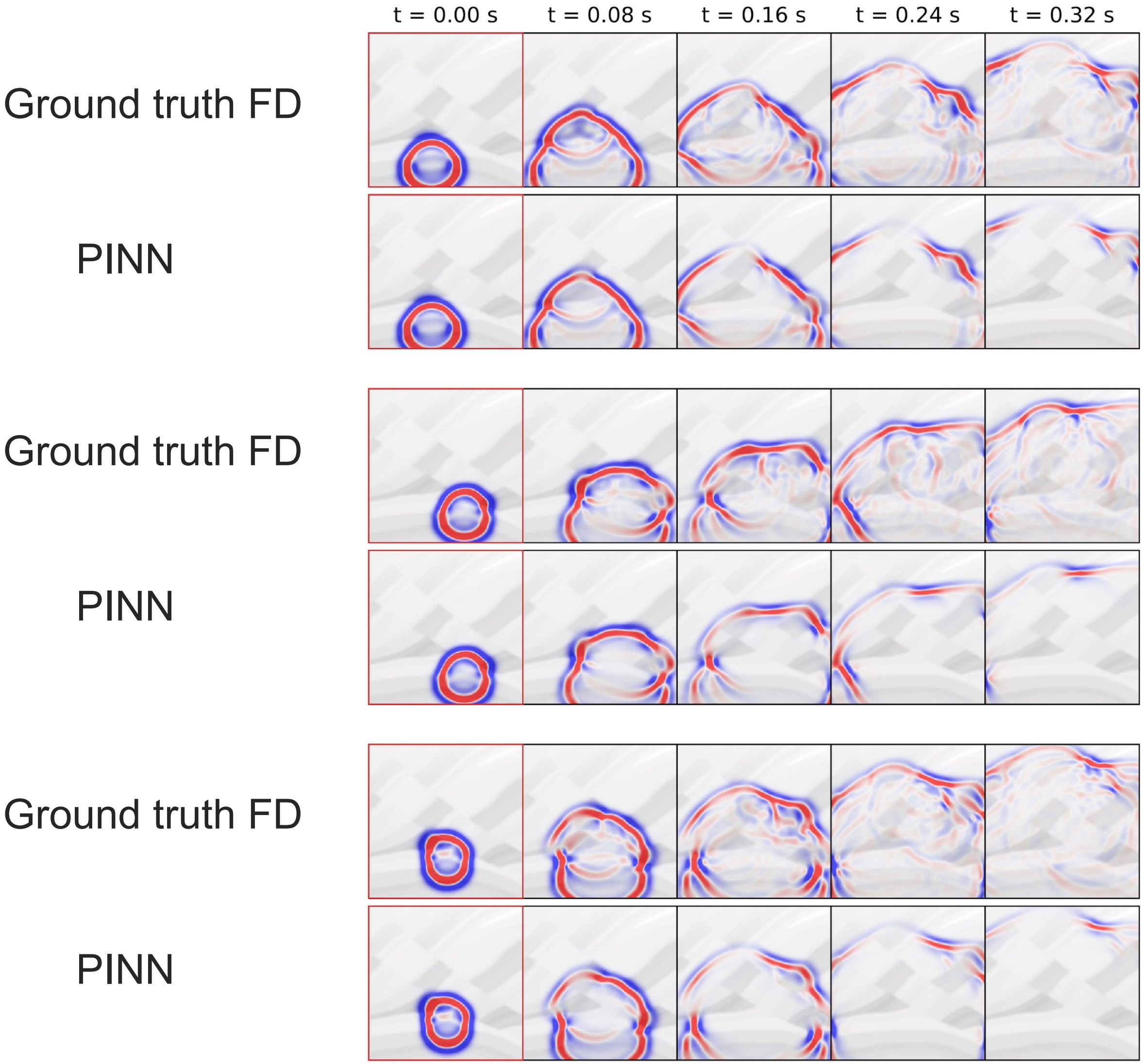
$$L_b(\theta) = \frac{\lambda}{N_b} \sum_{j}^{N_b} \Big(NN(x_j, t_j, s_j; \theta) - \underline{u_{FD}(x_j, t_j, s_j)} \Big)^2$$

$$L_p(\theta) = \frac{1}{N_p} \sum_{i}^{N_p} \left(\left[\nabla^2 - \frac{1}{c(x_i)^2} \frac{\partial^2}{\partial t^2} \right] NN(\underline{x_i, t_i, s_i}; \theta) \right)^2$$

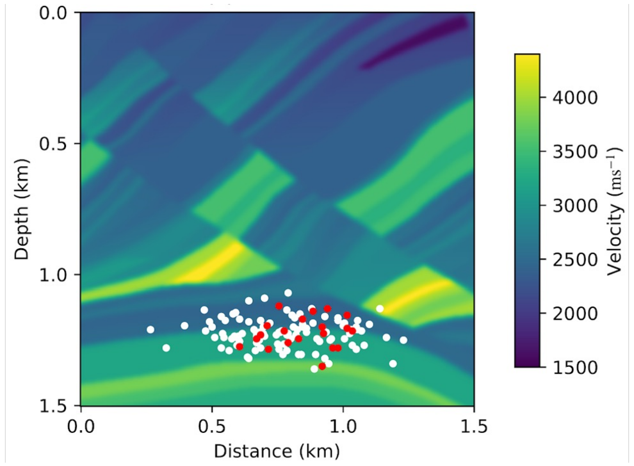
Randomly sampled input coordinates **and** source locations

Conditioned PINN

Conditioned PINNs

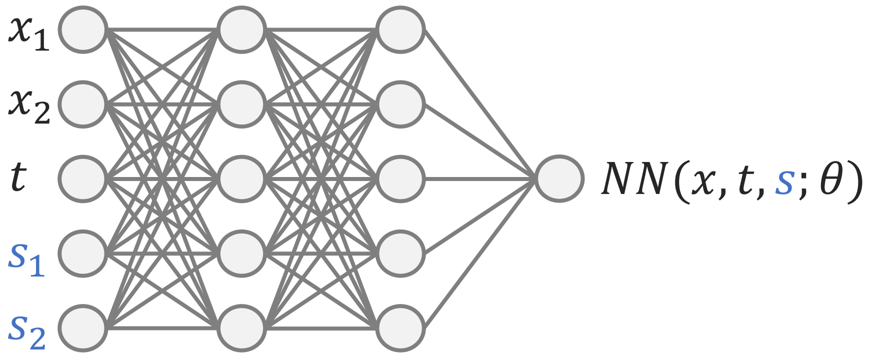


Velocity model, $c(x)$



White = random source locations used for training

Red = source locations used for testing



Means the network **does not** need to be retrained for each simulation => much faster!

Aka a **surrogate** model

Moseley et al, Solving the wave equation with physics-informed deep learning, ArXiv (2020)

Physics-informed deep operator networks (DeepONets)

Conditioned PINN for solving reaction-diffusion equation:

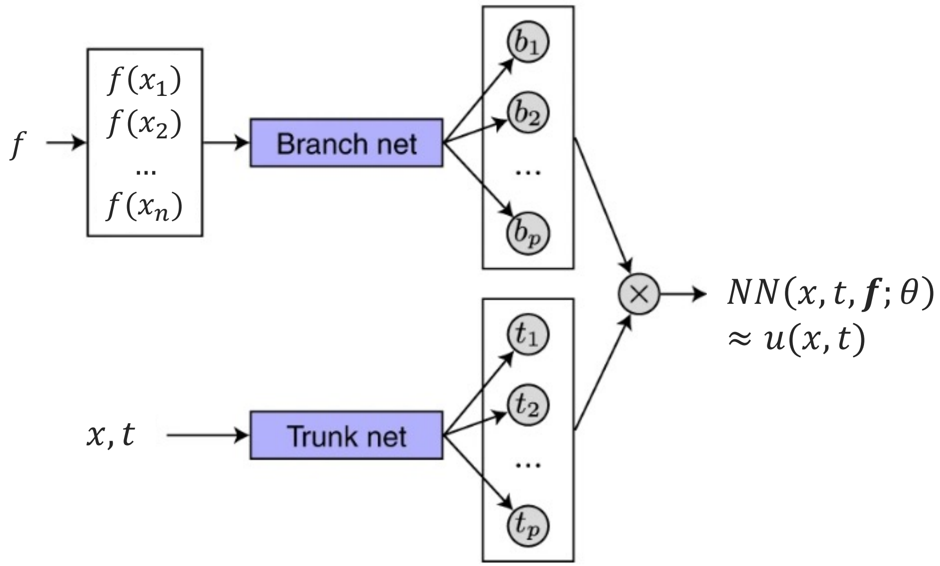
$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ku^2 + f(x)$$

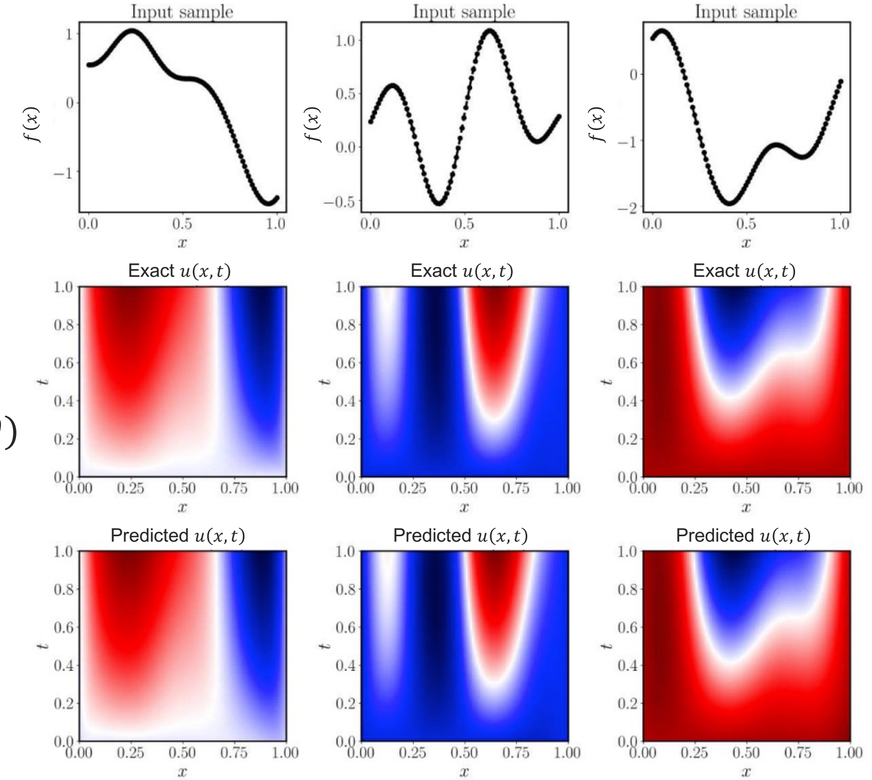
Input:
n discretised values of $f(x)$

Output:
$$NN(x,t,\boldsymbol{f};\theta) \approx u(x,t)$$

Trained using many examples of $f(x)$

DeepONet





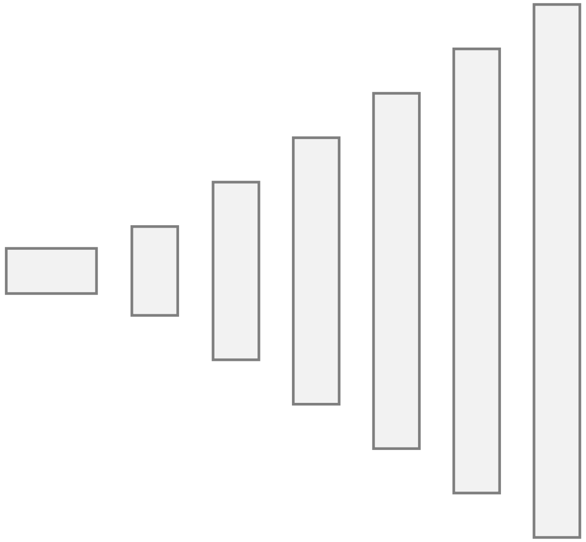
Wang et al, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Science Advances (2021)

Lu et al, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nature Machine Intelligence (2021)

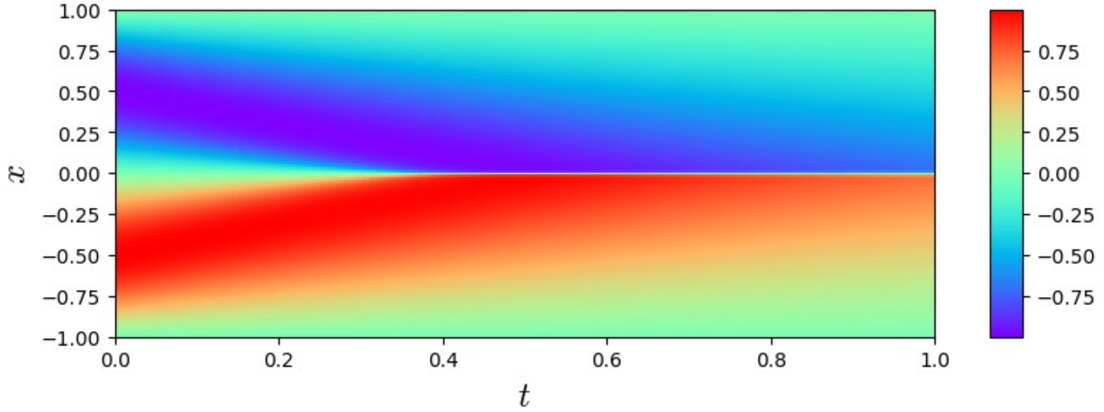
Discretised PINNs

Idea: **discretise** solution and use e.g. convolutional network to learn spatial/temporal correlations

Input: 0
(fixed value)



Architecture:
(Transposed) convolutional layers



Output: **discretised** solution
(a matrix of values)
e.g. 128 x 256

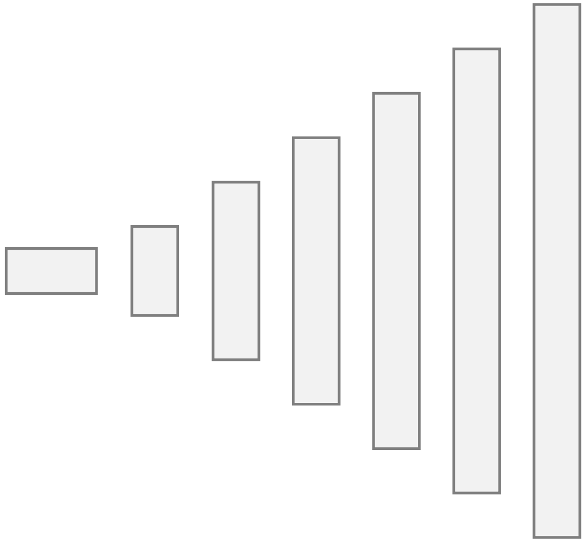
$$NN(\theta)_{ij} \approx u(x = x_i, t = t_j)$$

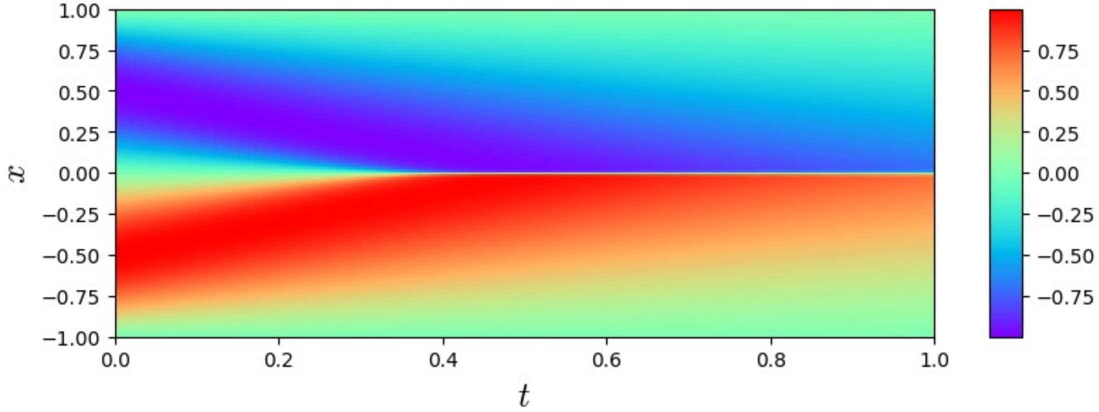
Instead of: $NN(x, t; \theta) \approx u(x, t)$

Discretised PINNs

Idea: **discretise** solution and use e.g. convolutional network to learn spatial/temporal correlations

Input: 0
(fixed value)





$$NN(x,t;\theta) \approx u(x,t)$$

$$L_p(\theta) = \frac{1}{N_p}\sum_{i}^{N_p}\left(\left(\frac{\partial NN}{\partial t} + NN\frac{\partial NN}{\partial x} - \nu\frac{\partial^2 NN}{\partial x^2}\right)(x_i,t_i;\theta)\right)^2$$

PINN

$$NN(\theta)_{ij} \approx u(x=x_i, t=t_j)$$

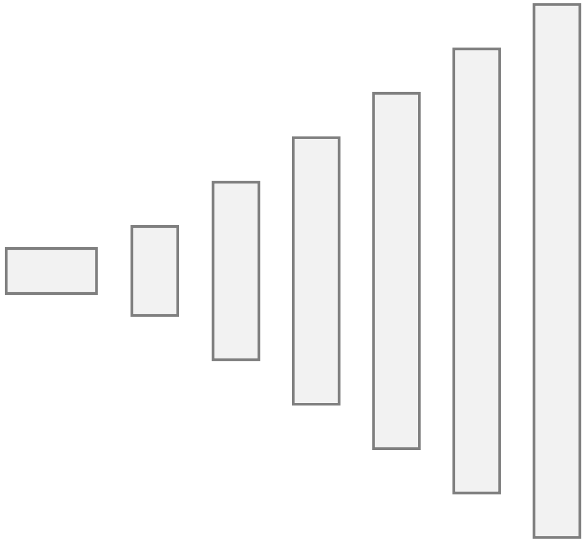
$$L_p(\theta) = \frac{1}{N_x N_t}\sum_{i}^{N_x}\sum_{j}^{N_t}\left(\frac{\delta NN(\theta)_{ij}}{\delta t} + NN(\theta)_{ij}\frac{\delta NN(\theta)_{ij}}{\delta x} - \nu\frac{\delta^2 NN(\theta)_{ij}}{\delta x^2}\right)^2$$

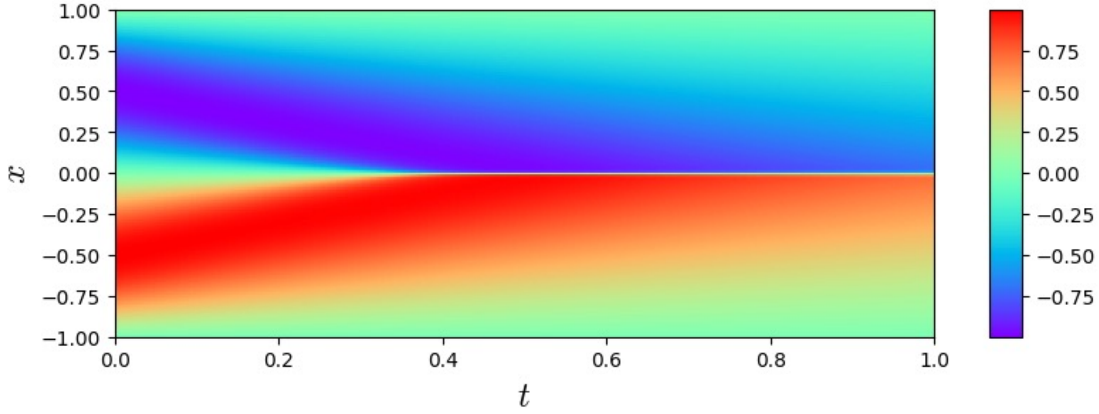
Discretised PINN

Discretised PINNs

Idea: **discretise** solution and use e.g. convolutional network to learn spatial/temporal correlations

Input: 0
(fixed value)





Derivatives in loss function are **approximated** using **finite difference** filters, e.g.

$$\frac{\delta NN(\theta)_{ij}}{\delta t} = \frac{NN(\theta)_{ij} - NN(\theta)_{i\,j-1}}{t_j - t_{j-1}}$$

Autodiff is still used to update θ

$$NN(\theta)_{ij} \approx u(x = x_i, t = t_j)$$

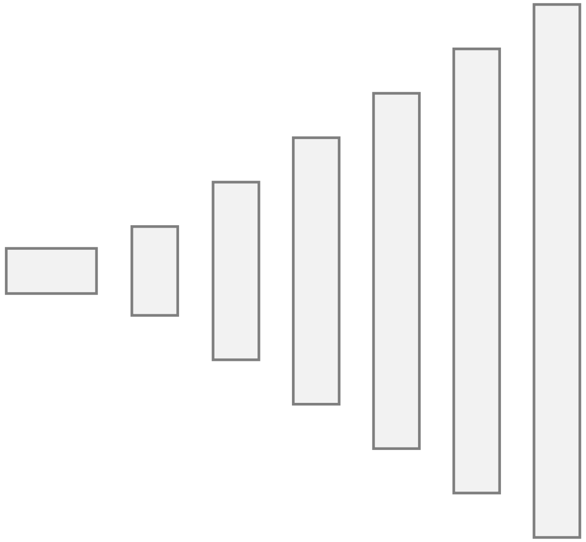
$$L_p(\theta) = \frac{1}{N_x N_t} \sum_i^{N_x} \sum_j^{N_t} \left(\frac{\delta NN(\theta)_{ij}}{\delta t} + NN(\theta)_{ij} \frac{\delta NN(\theta)_{ij}}{\delta x} - \nu \frac{\delta^2 NN(\theta)_{ij}}{\delta x^2} \right)^2$$

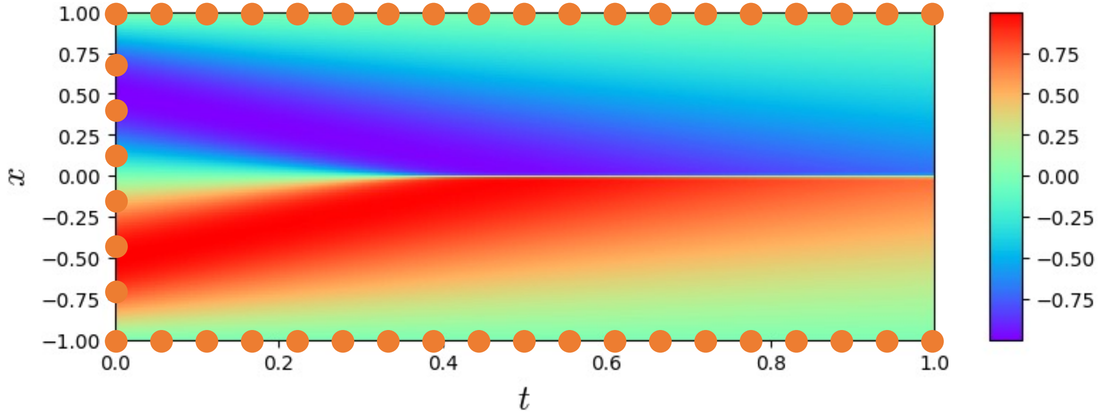
Discretised PINN

Discretised PINNs

Idea: **discretise** solution and use e.g. convolutional network to learn spatial/temporal correlations

Input: 0
(fixed value)





Initial/boundary conditions are asserted by **padding** the edges of the output solution with **appropriate** values (=hard constraint), e.g.

$$NN(\theta)_{i\ j=0} = \underline{-\sin(\pi x_i)}$$
$$NN(\theta)_{i=0\ j} = NN(\theta)_{i=128\ j} = \underline{0}$$

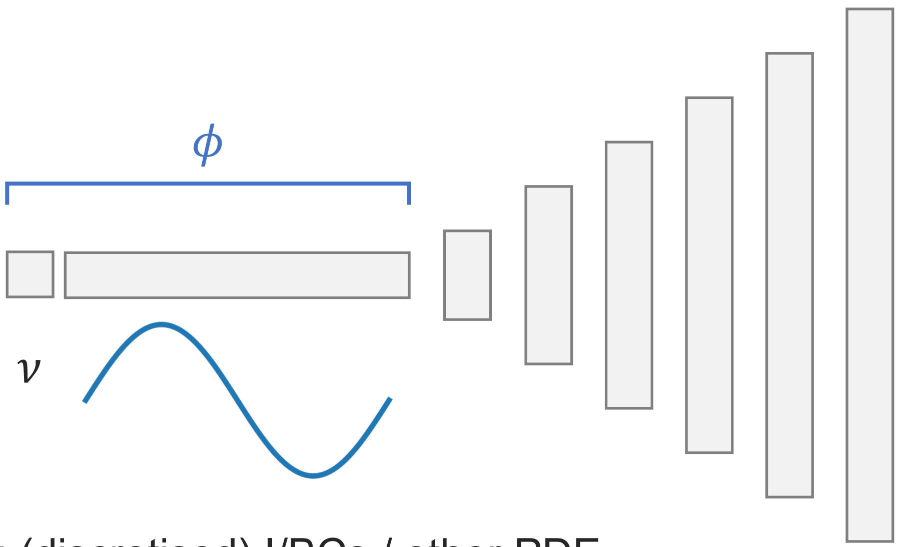
$$NN(\theta)_{ij} \approx u(x = x_i, t = t_j)$$

$$L_p(\theta) = \frac{1}{N_x N_t} \sum_i^{N_x} \sum_j^{N_t} \left(\frac{\delta NN(\theta)_{ij}}{\delta t} + NN(\theta)_{ij} \frac{\delta NN(\theta)_{ij}}{\delta x} - \nu \frac{\delta^2 NN(\theta)_{ij}}{\delta x^2} \right)^2$$

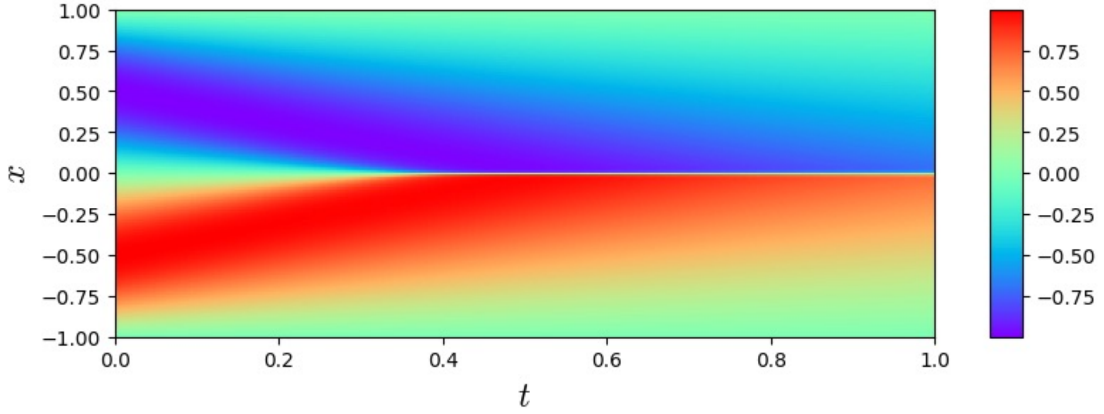
Discretised PINN

Conditioned discretised PINNs

Idea: **discretise** solution and use e.g. convolutional network to learn spatial/temporal correlations
And condition on I/BCs / other PDE parameters



Input: (discretised) I/BCs / other PDE parameters



$$NN(\phi;\theta)_{ij} \approx u(x = x_i, t = t_j)$$

$$L_p(\theta) = \frac{1}{N_x N_t N_\phi} \sum_{k}^{N_\phi} \sum_{i}^{N_x} \sum_{j}^{N_t} \left(\frac{\delta NN(\phi_k;\theta)_{ij}}{\delta t} + NN(\phi_k;\theta)_{ij} \frac{\delta NN(\phi_k;\theta)_{ij}}{\delta x} - \phi_{k0} \frac{\delta^2 NN(\phi_k;\theta)_{ij}}{\delta x^2} \right)^2$$

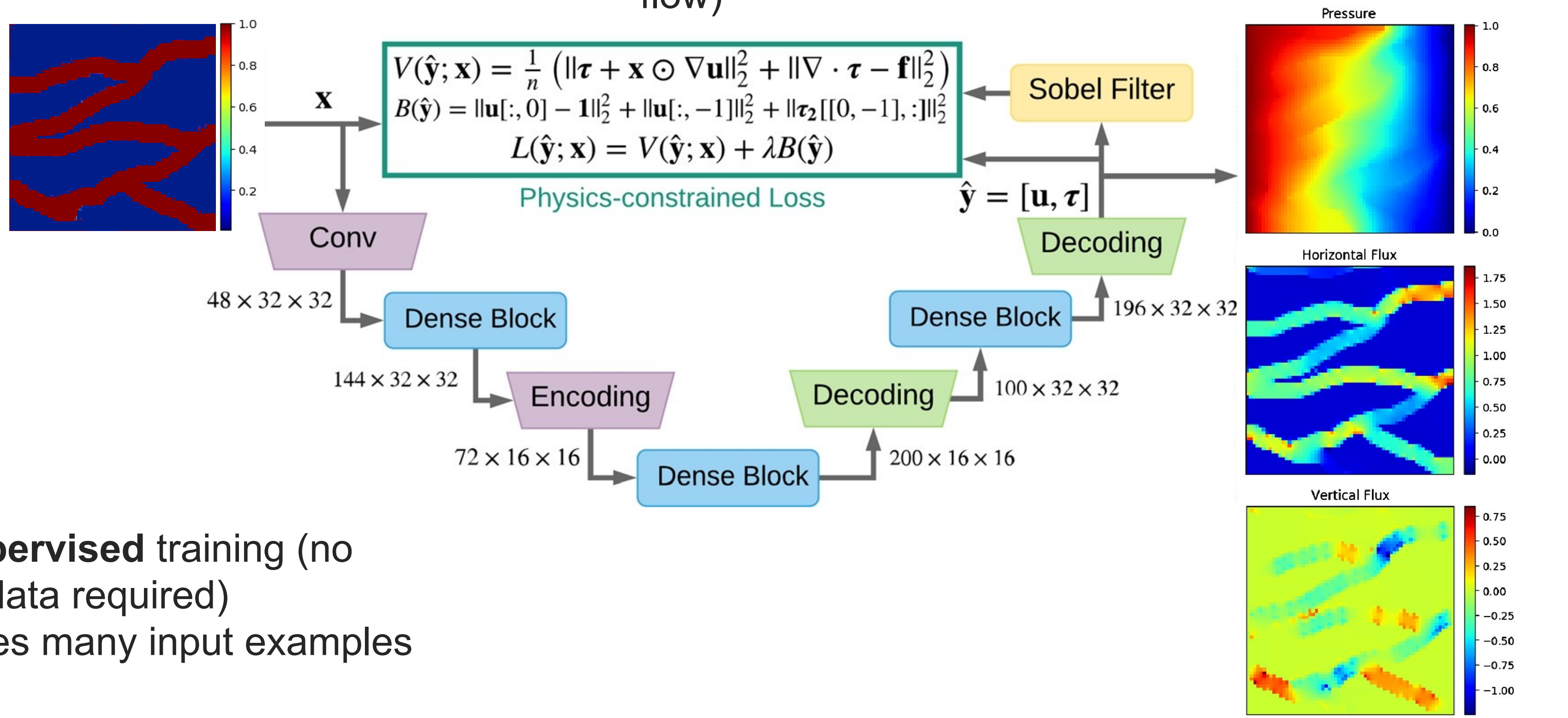
Conditioned discretised PINN

Conditioned discretised PINNs

Input: permeability field

PDE: steady-state flow in heterogeneous media (Darcy flow)

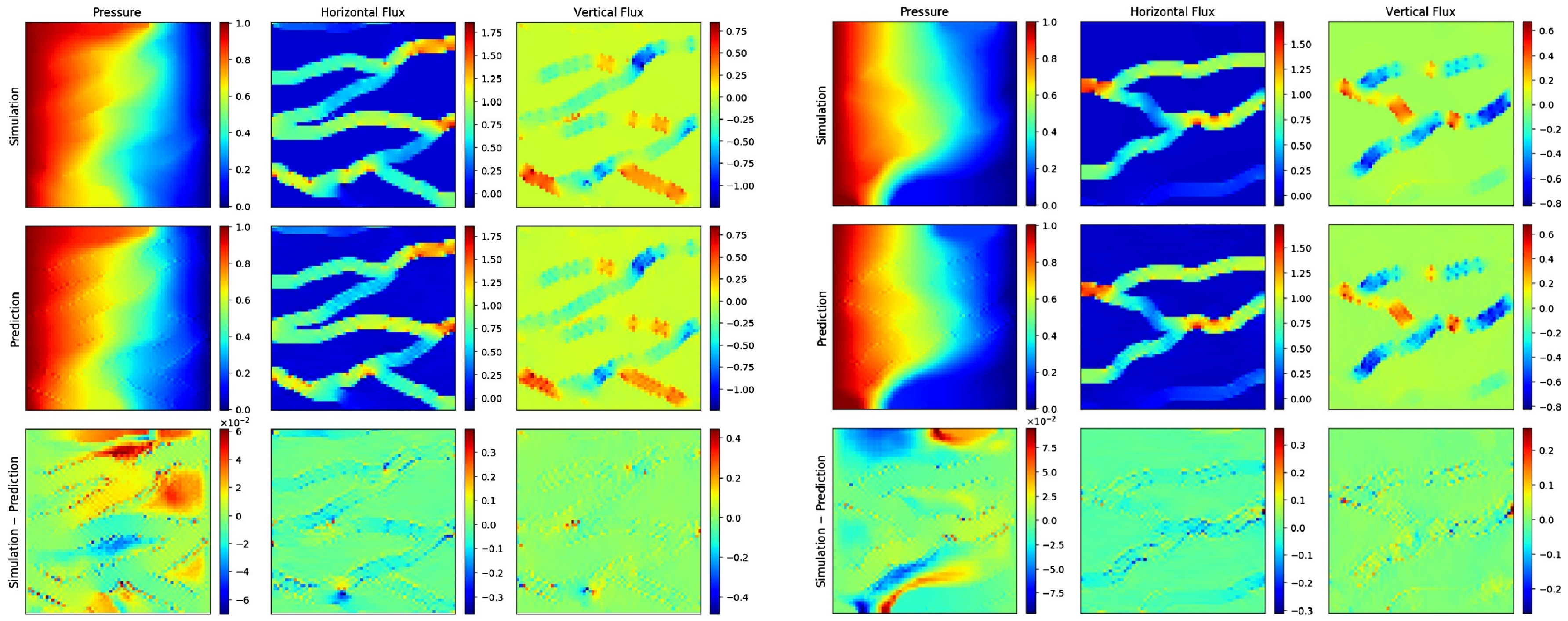
Output: fluid pressure and flux field



- Fully **unsupervised** training (no simulation data required)
- Only requires many input examples to train

Zhu, Y et al, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics (2019)

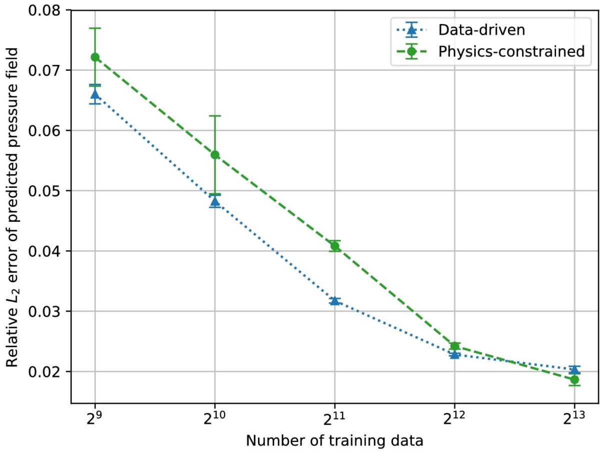
Conditioned discretised PINNs



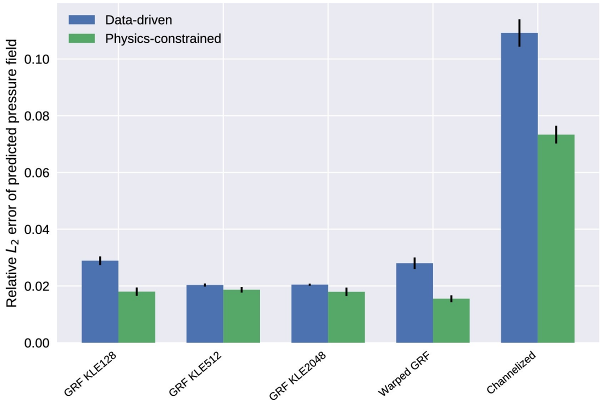
(c) Channelized, test 1.

(d) Channelized, test 2.

Physics-informed vs fully data-driven CNN



Generalisation to different input distributions



Zhu, Y et al, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics (2019)

Advantages / limitations of discretised PINNs

Advantages

- Allows the use of CNNs to exploit spatial correlations between inputs/outputs of PDE
- Can be extended to:
  - Irregular geometries (Graph NNs)
  - Explicit time dependence (RNNs)
  - Mixed continuous/discrete input coordinates

Limitations

- Relies on approximate FD gradients
- Only outputs discretised solution; needs to be retrained to output on larger domains / finer grids

Geneva et al, Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks, JCP (2020)

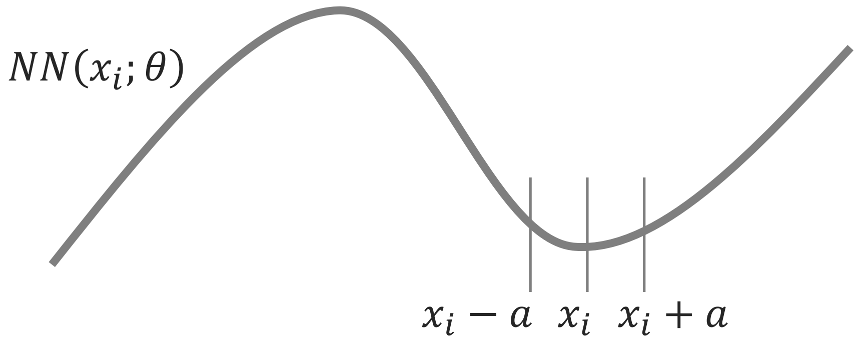
Gao et al, PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, JCP (2021)

Training PINNs with finite differences

Most time is spent **computing gradients**, not the forward pass, when training PINNs

$$L_p(\theta) = \frac{1}{N_p}\sum_{i}^{N_p} \|\mathcal{D}[NN(x_i;\theta)] - f(x_i)\|^2$$

Idea: instead of using exact gradients from autodifferentiation, use **approximate** gradients from **finite differences**



$$\frac{\partial NN(x_i;\theta)}{\partial x} \approx \frac{NN(x_i + a;\theta) - NN(x_i - a;\theta)}{2a}$$

Sharma et al, Accelerated Training of Physics-Informed Neural Networks (PINNs) using Meshless Discretizations, NeurIPS (2022)

Training PINNs with finite differences

Most time is spent **computing gradients**, not the forward pass, when training PINNs

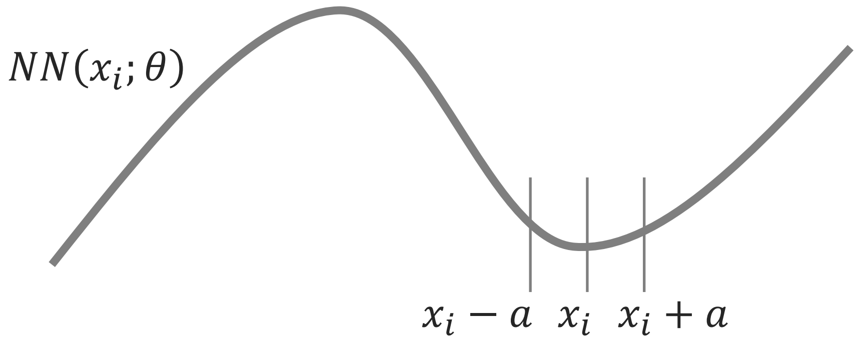
$$L_p(\theta) = \frac{1}{N_p} \sum_{i}^{N_p} \| \mathcal{D}[NN(x_i;\theta)] - f(x_i) \|^2$$

Idea: instead of using exact gradients from autodifferentiation, use **approximate** gradients from **finite differences**

For each collocation point x_i:

1. Sample a stencil of input points around x_i
2. Run forward pass of network with all these points
3. Approximate derivatives in $\mathcal{D}$ using finite differences
4. Compute loss function using these derivatives

Autodiff is still used to update θ



$$\frac{\partial NN(x_i;\theta)}{\partial x} \approx \frac{NN(x_i + a;\theta) - NN(x_i - a;\theta)}{2a}$$

Can offer 2-4x speedups depending on PDE

But choosing a suitable value of a is critical

Sharma et al, Accelerated Training of Physics-Informed Neural Networks (PINNs) using Meshless Discretizations, NeurIPS (2022)

Overview of lectures

- PINN limitations
- PINN extensions for improving:
  - Computational cost
  - Convergence / accuracy
  - Scalability to more complex problems
- Summary: when should I use PINNs?

Part 1

Part 2

Learning objectives

- Explain the advantages and disadvantages of PINNs
- Understand current research directions on improving their performance

Lecture summary

- Standard PINNs suffer from some major limitations:
  - Computational cost
  - Poor convergence
  - Scaling to more complex problems
- We can improve their computational cost by:
  - Conditioning them on additional inputs
  - Discretising them
  - Using finite differences to train them