Deep Learning in Scientific Computing

PINNs – limitations and extensions

Spring Semester 2023

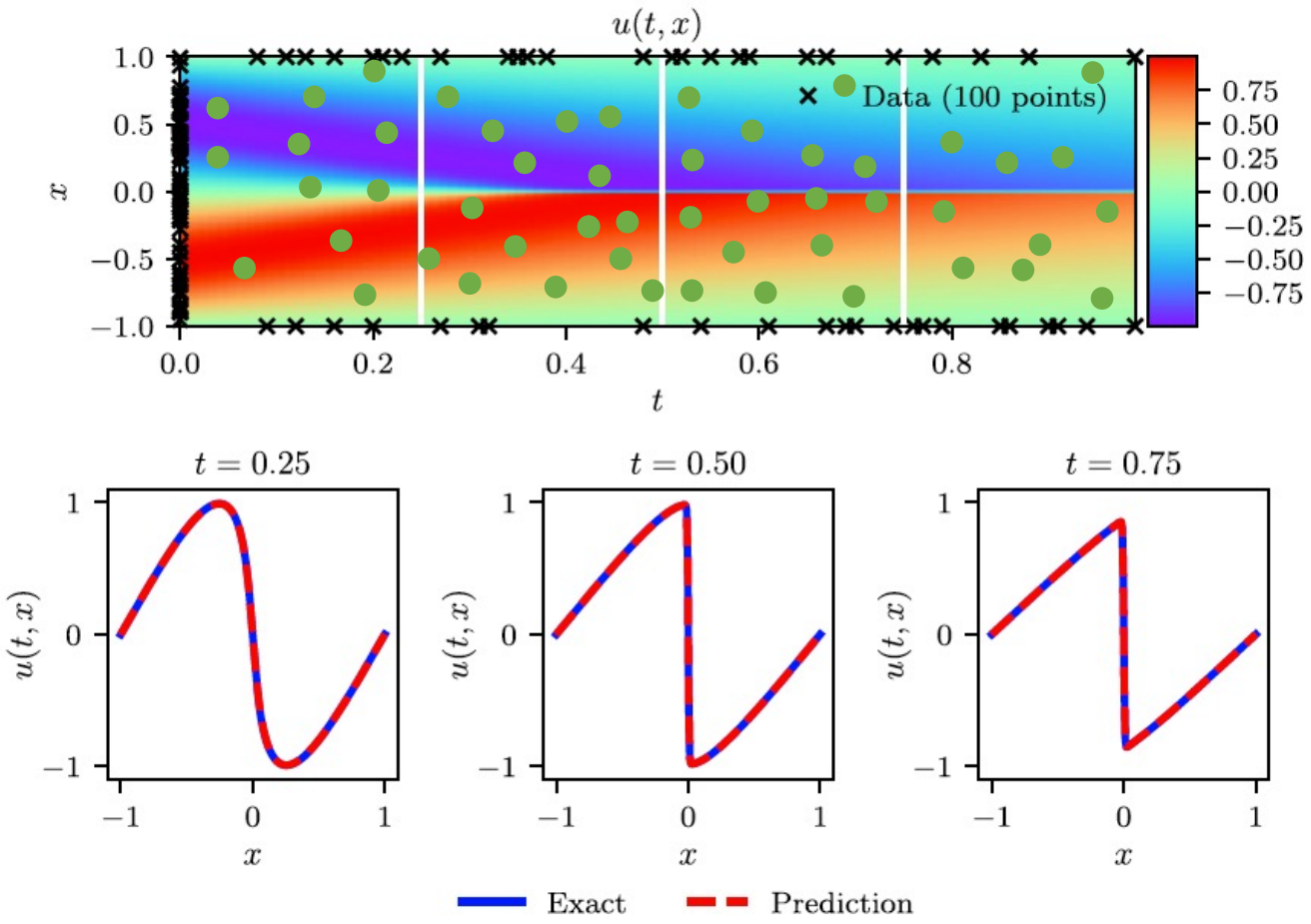
Siddhartha Mishra
Ben Moseley

ETH zürich

Course timeline

Tutorials		Lectures	
Tue 3:15-14:00, HG E5		Fri 12:15-14:00, HG D1.1	
21.02.		24.02.	~~Course introduction~~
28.02.	~~Intro to PyTorch~~	03.03.	~~Introduction to deep learning I~~
07.03.	~~Simple DNNs in PyTorch~~	10.03.	~~Introduction to deep learning II~~
14.03.	~~Advanced DNNs in PyTorch~~	17.03.	~~Physics-informed neural networks – introduction and theory~~
21.03.	~~PINN exercises~~	24.03.	~~Physics-informed neural networks – applications~~
28.03.	~~Implementing PINNs I~~	31.03.	**Physics-informed neural networks – limitations and extensions**
04.04.	Implementing PINNs II	07.04.	
11.04.		14.04.	
18.04.	Introduction to projects	21.04.	Neural operators – introduction and theory
25.04.	Implementing neural operators I	28.04.	Neural operators – applications
02.05.	Implementing neural operators II	05.05.	Neural operators – extensions
09.05.	Operator learning exercises	12.05.	Graph and sequence models
16.05.	Project discussions	19.05.	Differentiable physics – introduction
23.05.	Implementing autodifferentiation	26.05.	Differentiable physics and neural differential equations
30.05.	Intro to JAX	02.06.	Future trends and overview of CAMLAB

Recap – PINNs



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

Solving Burgers' equation:

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

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

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

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

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

Recap – PINN training loop

Training loop:

1. Sample boundary/ physics training points
2. Compute network outputs
3. Compute 1st and 2nd order gradient of network output **with respect to network input <= (recursively) apply autodiff, extending graph**
4. Compute loss
5. Compute gradient of loss function **with respect to network parameters <= apply autodiff on extended graph**
6. Take gradient descent step

We can recursively apply **autodifferentiation** to compute gradients and extend the graph

```
# PINN training psuedocode

#2.
t.requires_grad_(True)# tells PyTorch to start tracking graph
theta.requires_grad_(True)
u = NN(t, theta)

#3.
dudt = torch.autograd.grad(u, t, torch.ones_like(u),
                           create_graph=True)[0]
d2udt2 = torch.autograd.grad(dudt, t, torch.ones_like(u),
                             create_graph=True)[0]

#4.
physics_loss = torch.mean((m*d2udt2 + mu*dudt + k*u)**2)
loss = physics_loss + lambda_*boundary_loss

#5.
dtheta = torch.autograd.grad(loss, theta)[0]
```

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**:

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

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

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

Lecture overview

- PINN applications: recap
- PINN limitations & extensions
  - Computational cost
  - Poor convergence
  - Scaling to more complex problems
- Summary: PINNs: when should I use them?

Advantages / limitations of PINNs

Advantages

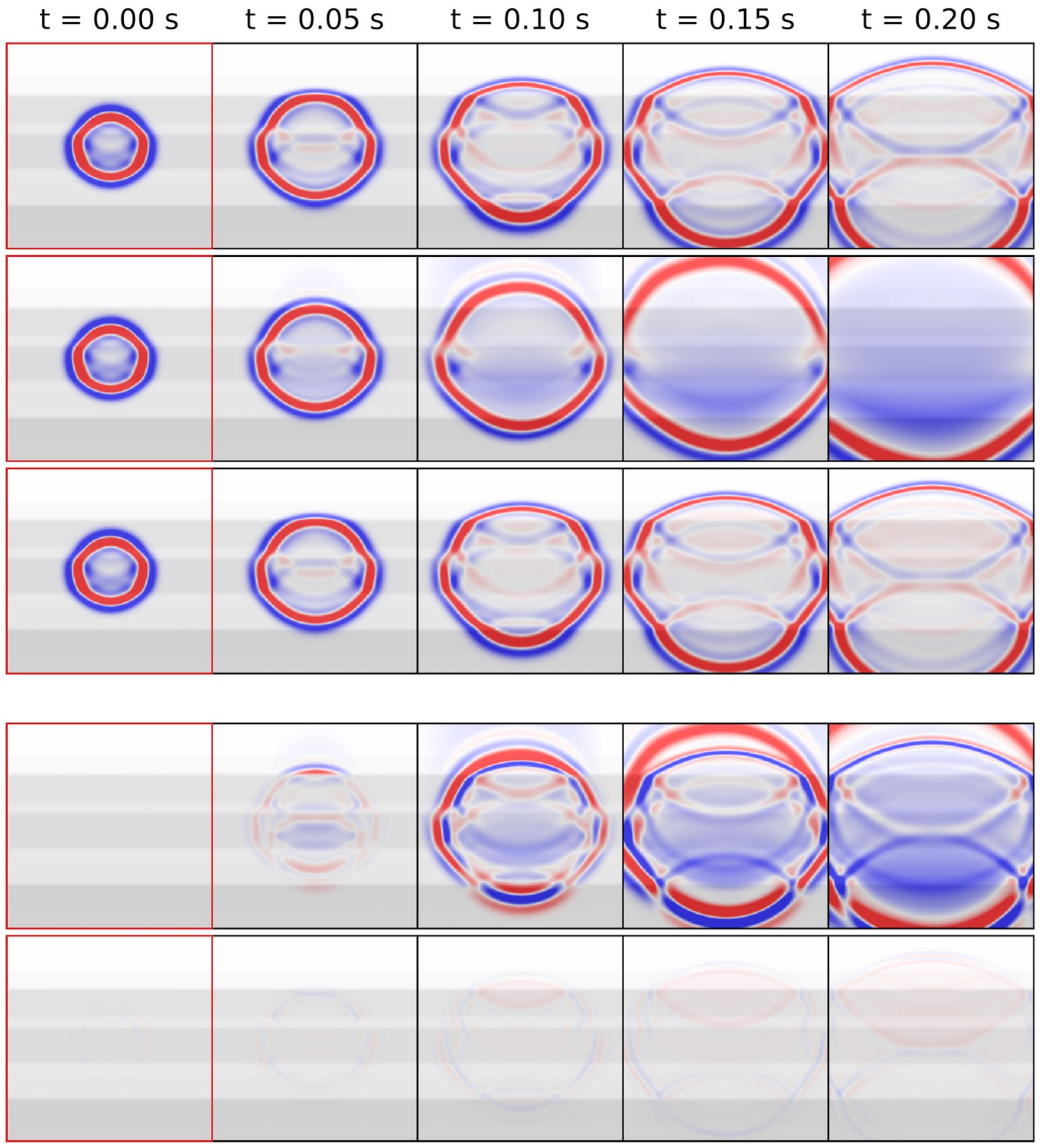
- **Mesh-free**
- Can perform well for high-dimensional PDEs
- Can be **extended** to solve inverse and discovery problems
- Often perform best on “messy/**mixed**” problems, where some noisy data is available, physics is perhaps not perfectly known, and traditional algorithms require many forward evaluations
- Tractable, analytical solution gradients (e.g. for sensitivity analysis)
- Mostly **unsupervised**

Limitations

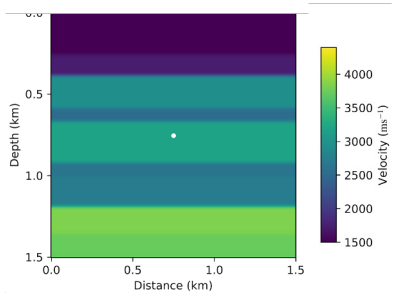
- **?**

PINN limitation 1) – computational cost

PINNs for solving wave equation



Velocity model, $c(x)$

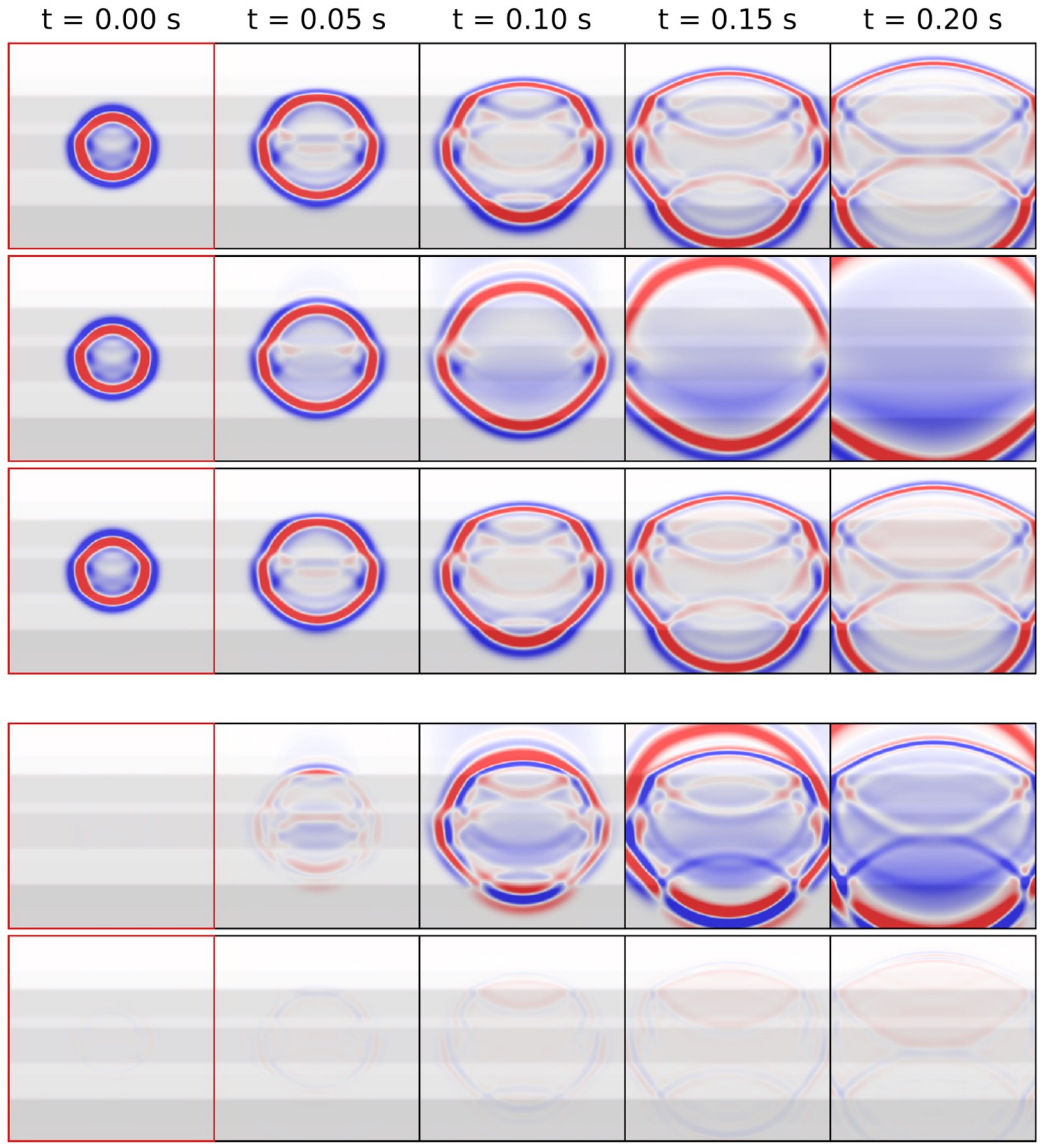


Moseley et al, Solving the wave equation with physics-informed deep learning, ArXiv (2020)

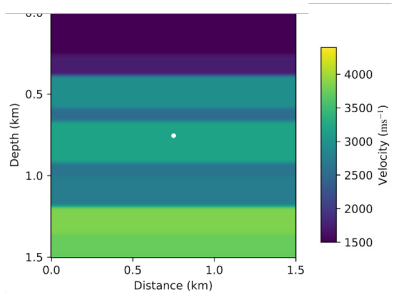
Mini-batch size $N_b = N_p = 500$ (random sampling)
Fully connected network with 10 layers, 1024 hidden units
Softplus activation
Adam optimiser

Training time: **~1 hour**

PINNs for solving wave equation



Velocity model, $c(x)$



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

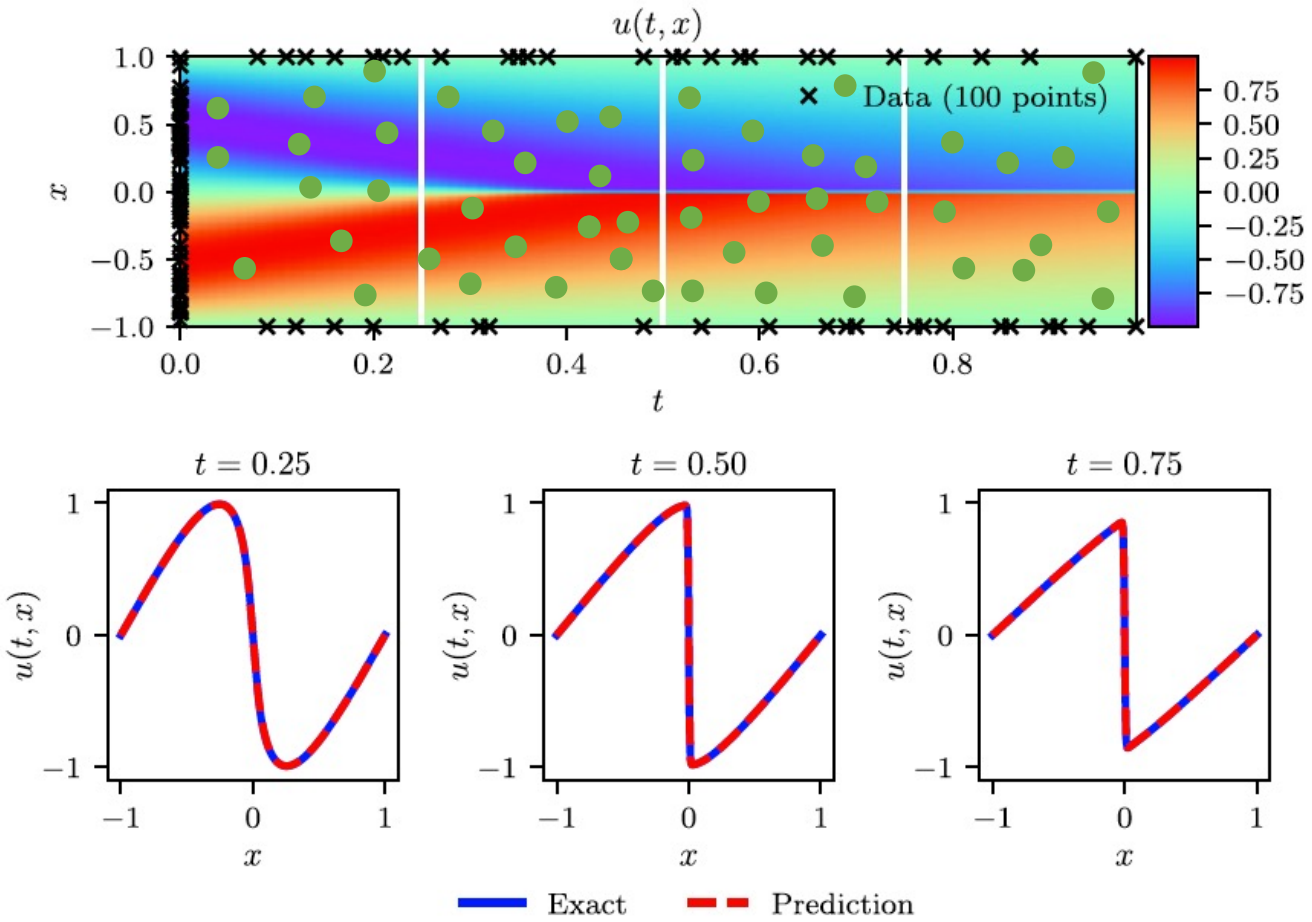
Mini-batch size $N_b = N_p = 500$ (random sampling)
Fully connected network with 10 layers, 1024 hidden units
Softplus activation
Adam optimiser

Training time: **~1 hour**

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

PINN limitation 2) – poor convergence

Competing loss terms

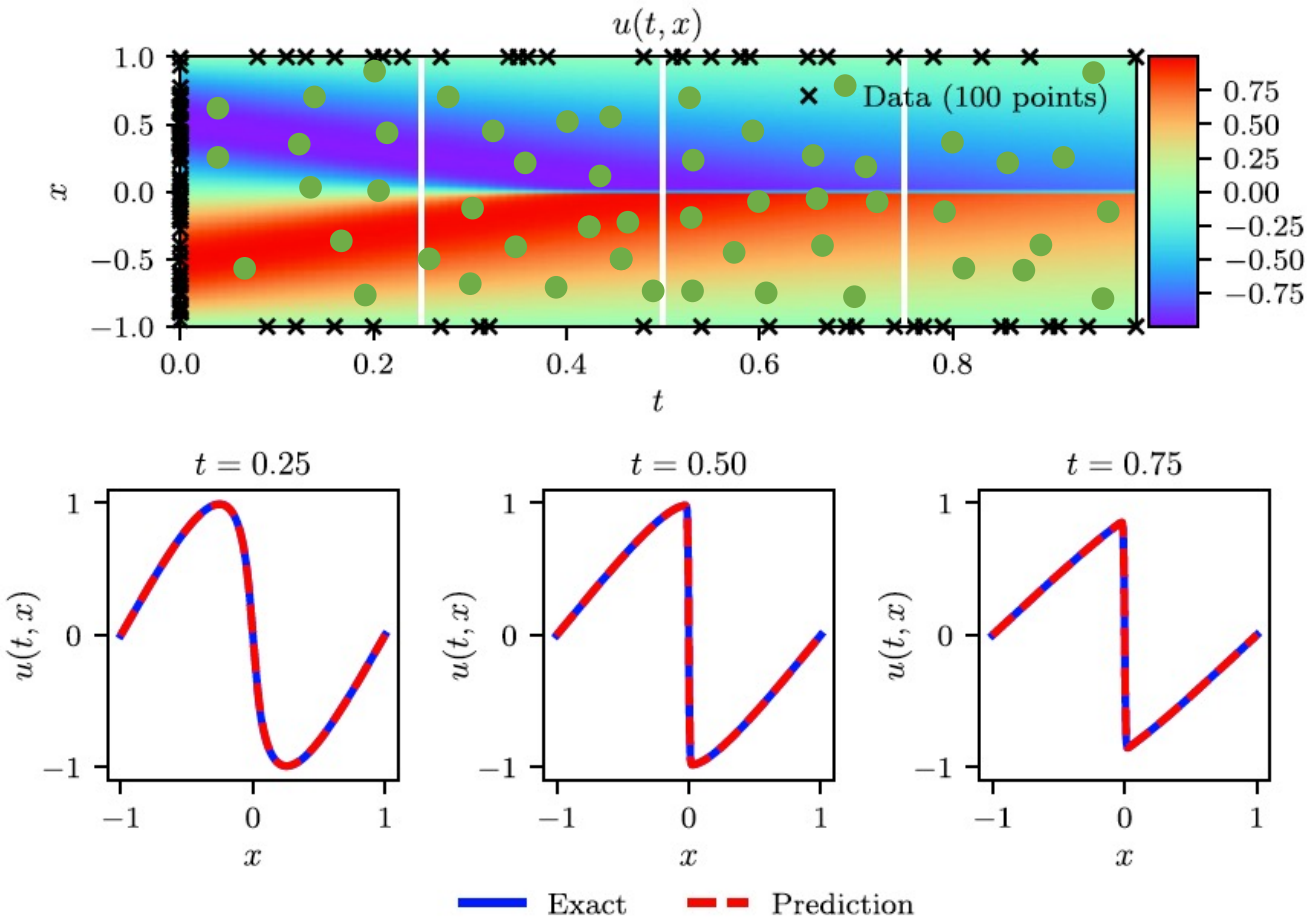


$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_j^{N_{b1}} \left(NN(x_j, 0; \theta) + \sin(\pi x_j)\right)^2 + \frac{\lambda_2}{N_{b2}} \sum_k^{N_{b2}} (NN(-1, t_k; \theta) - 0)^2 + \frac{\lambda_3}{N_{b3}} \sum_l^{N_{b3}} (NN(+1, t_l; \theta) - 0)^2$$

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

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



$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_j^{N_{b1}} \left(NN(x_j, 0; \theta) + \sin(\pi x_j) \right)^2$$
$$+ \frac{\lambda_2}{N_{b2}} \sum_k^{N_{b2}} (NN(-1, t_k; \theta) - 0)^2$$
$$+ \frac{\lambda_3}{N_{b3}} \sum_l^{N_{b3}} (NN(+1, t_l; \theta) - 0)^2$$
$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \left(\left(\frac{\partial NN}{\partial t} + NN \frac{\partial NN}{\partial x} - \nu \frac{\partial^2 NN}{\partial x^2} \right) (x_i, t_i; \theta) \right)^2$$

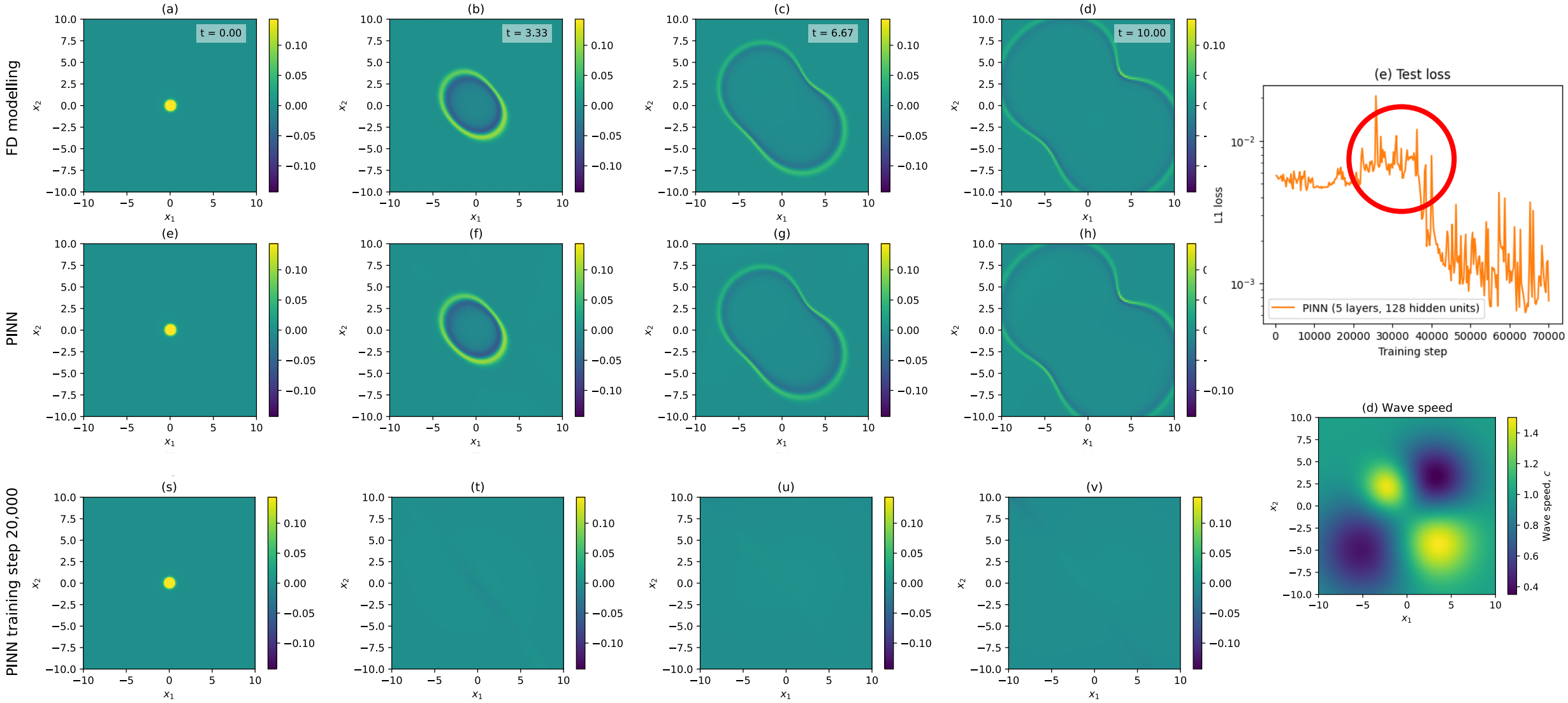
How do we choose λ_1, λ_2, and λ_3?

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

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

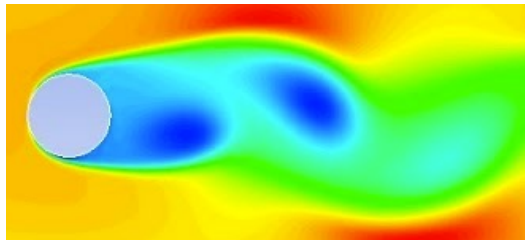
PINN solution “thrashing”

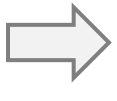


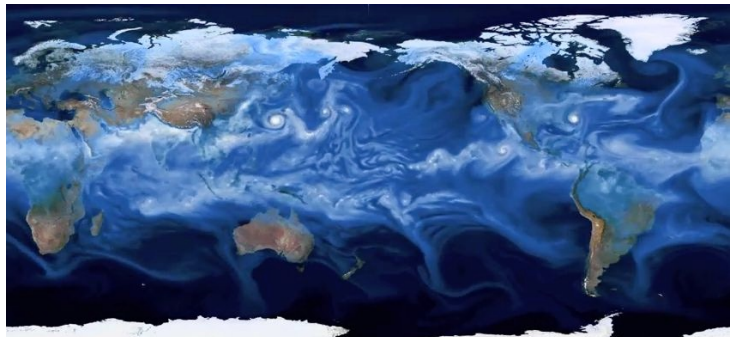
Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ArXiv (2021)

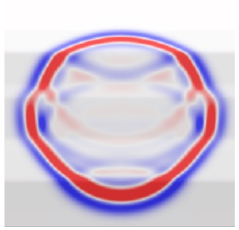
PINN limitation 3) – scaling to more complex problems

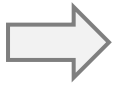
Scaling to more complex problems

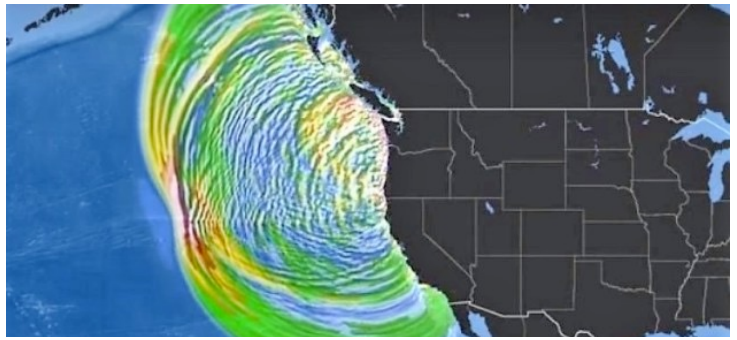








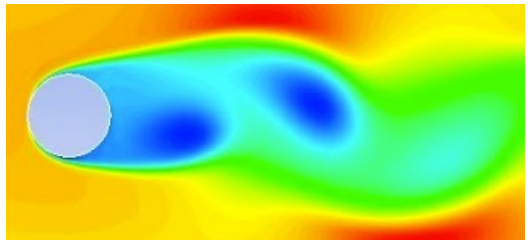


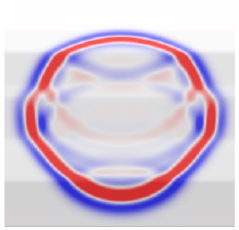


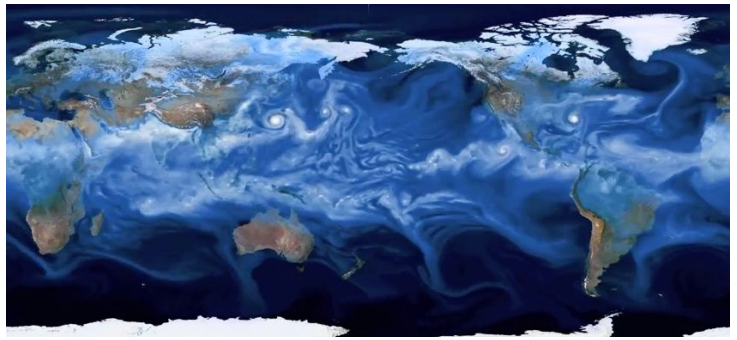
Most PINN research focuses on **toy/simplified** problems,
as proof-of-principle studies

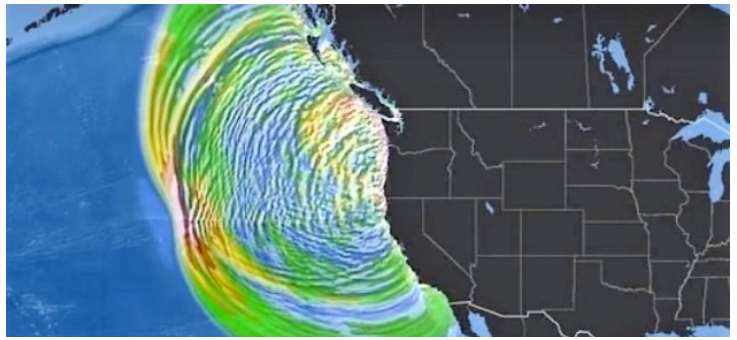
Image credits: Lawrence Berkeley National Laboratory / NOAA / NWS / Pacific Tsunami Warning Center

Scaling to more complex problems









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

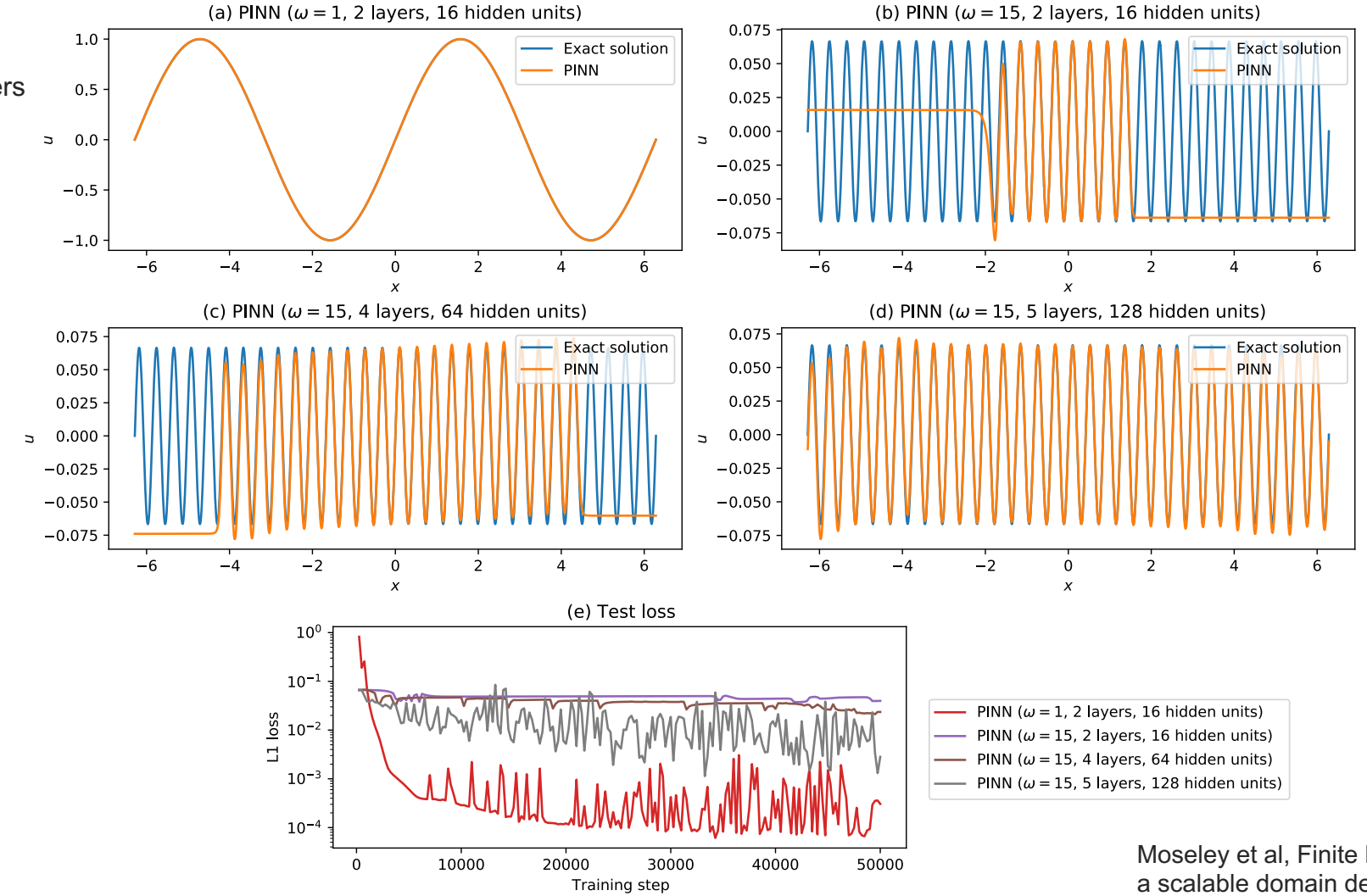
How do PINNs cope in this setting?

Most 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 PINNs to higher frequencies

321 free parameters



PINN solving:

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

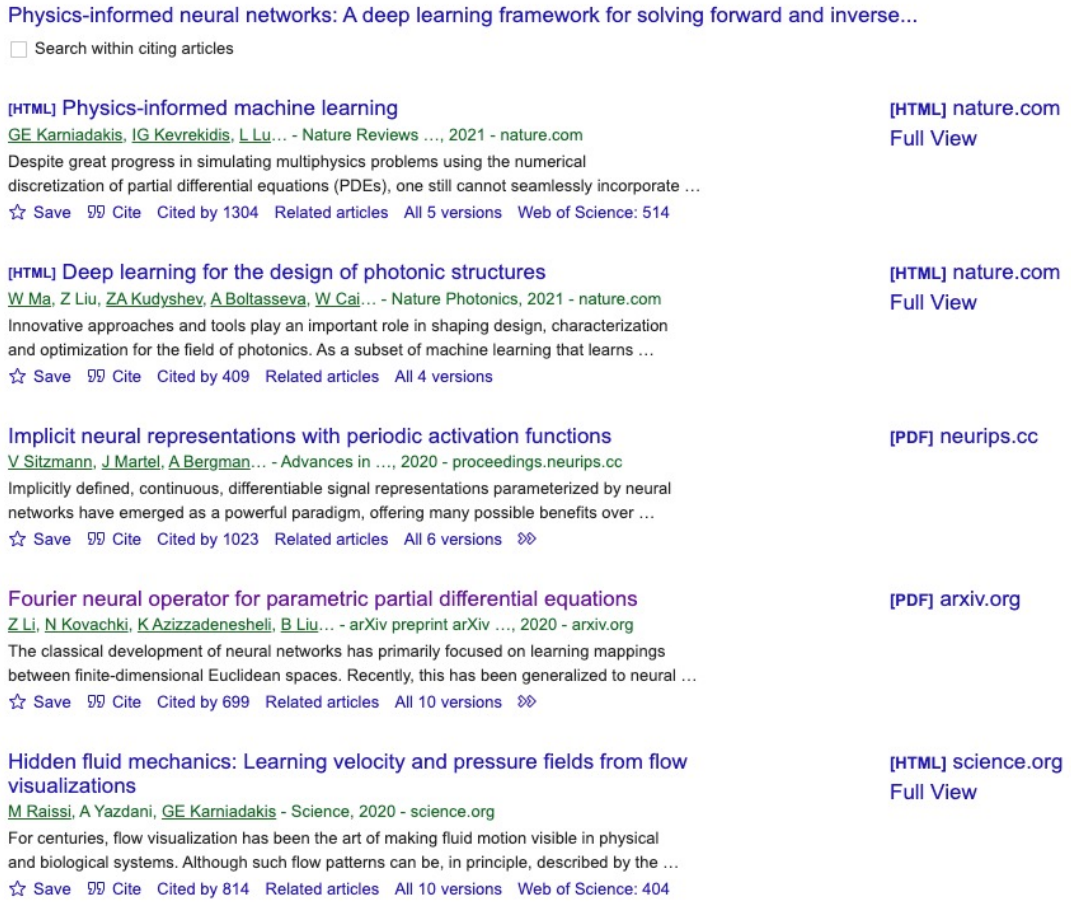
66,433 free parameters

Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ArXiv (2021)

PINN research landscape

Physics-informed neural networks # citations
Currently ~ 4700+

Cuomo et al, Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next, ArXiv (2022)



100-1000s of PINN applications / extensions!

Lecture overview

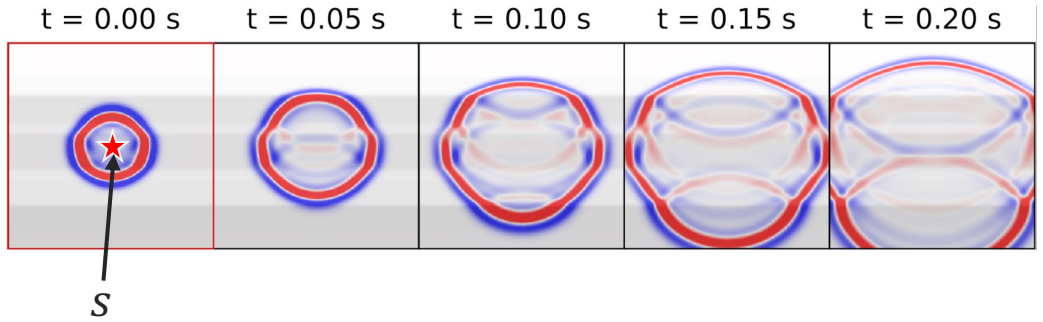
- PINN applications: recap
- PINN limitations & extensions
  - Computational cost
  - Poor convergence
  - Scaling to more complex problems
- Summary: PINNs: when should I use them?

Limitation 1) – computational cost

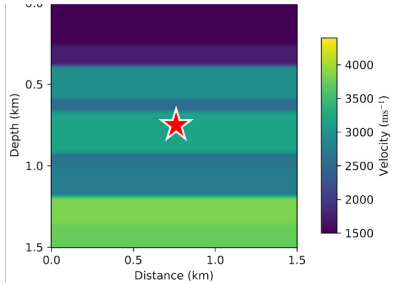
Conditioned PINNs

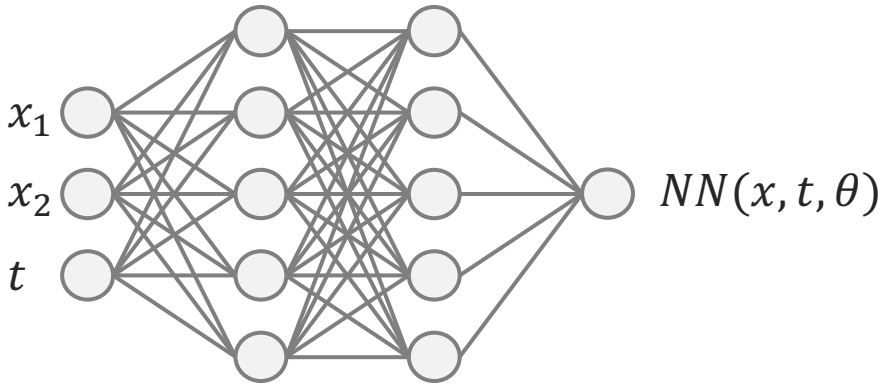
Idea: add I/BCs / other PDE parameters as **additional** network input parameters

Ground truth FD simulation



Velocity model, $c(x)$



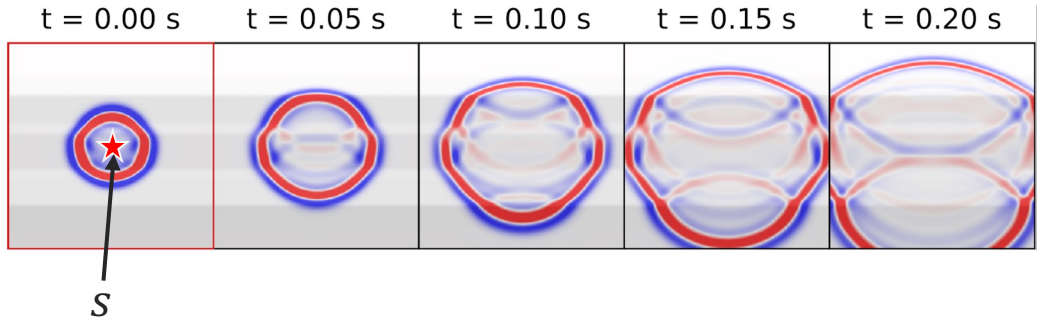


PINN

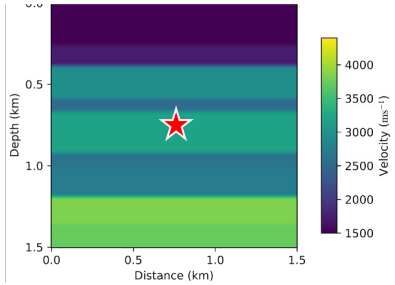
Conditioned PINNs

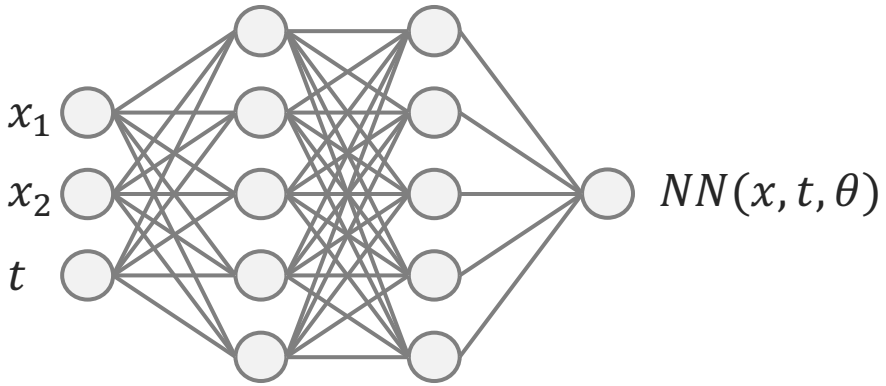
Idea: add I/BCs / other PDE parameters as **additional** network input parameters

Ground truth FD simulation

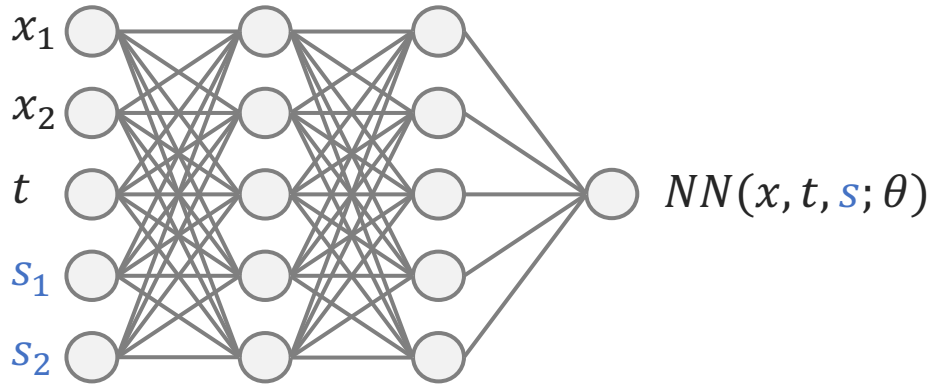


Velocity model, $c(x)$





PINN

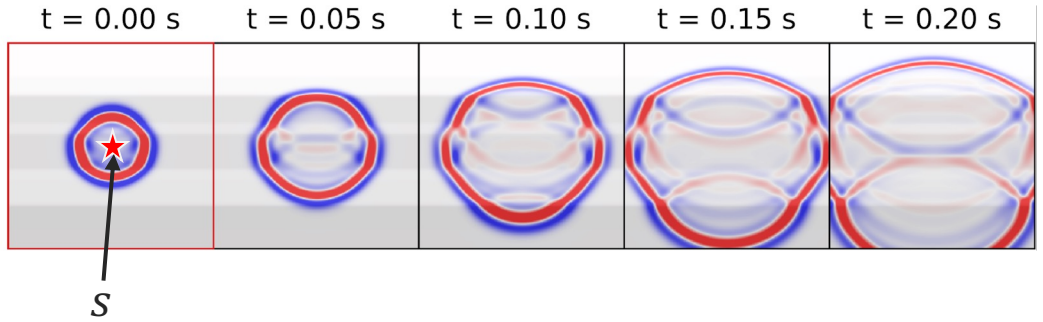


Conditioned PINN

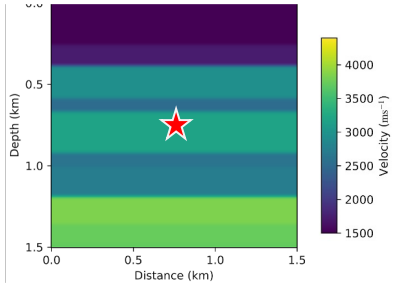
Conditioned PINNs

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

Ground truth FD simulation



Velocity model, $c(x)$



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

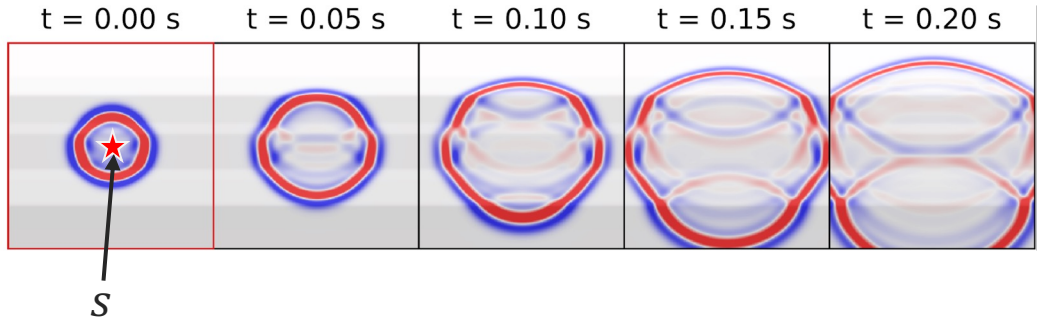
$$L_p(\theta) = \frac{1}{N_p}\sum_{i}^{N_p}\left(\left[\nabla^2 - \frac{1}{c(x_i)^2}\frac{\partial^2}{\partial t^2}\right] NN(x_i, t_i; \theta)\right)^2$$

PINN

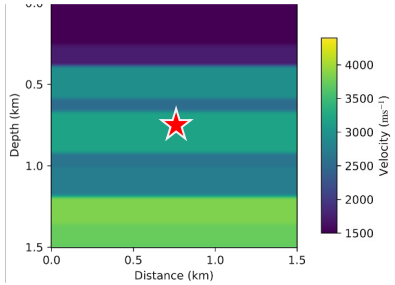
Conditioned PINNs

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

Ground truth FD simulation



Velocity model, $c(x)$



$$L_b(\theta) = \frac{\lambda}{N_b}\sum_j^{N_b}\left(NN(x_j,t_j;\theta) - u_{FD}(x_j,t_j)\right)^2$$

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

PINN

Boundary examples from many **different** source locations

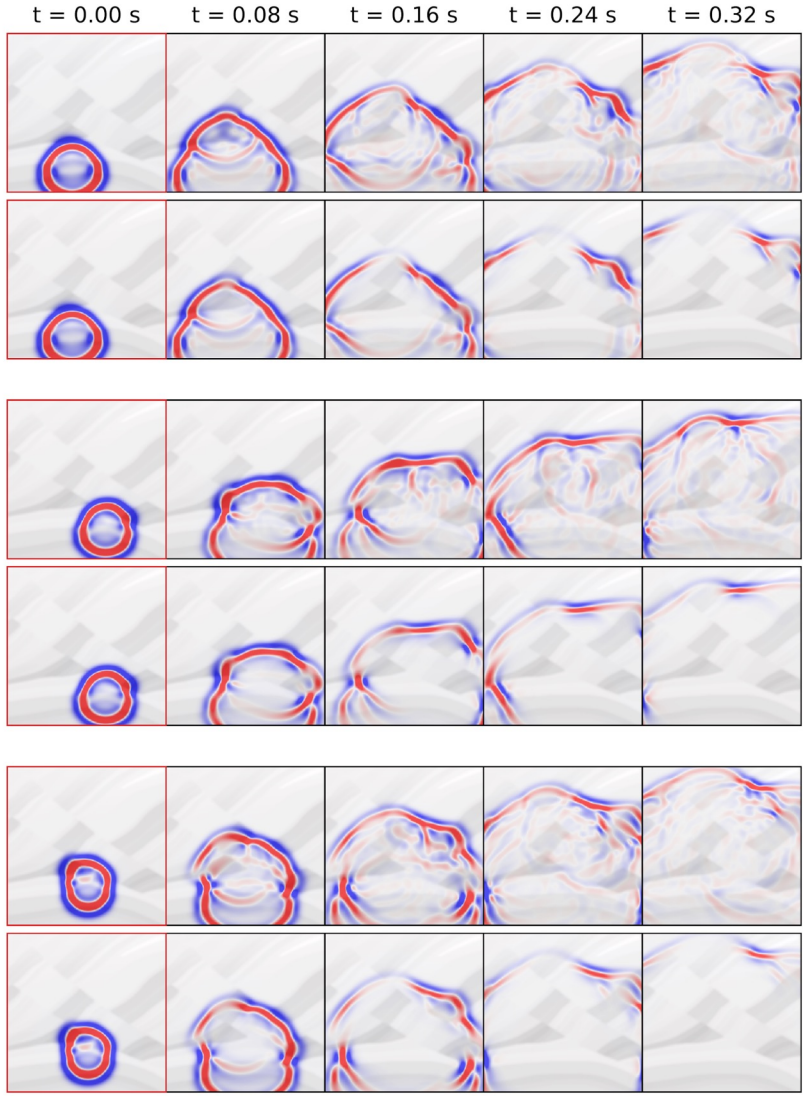
$$L_b(\theta) = \frac{\lambda}{N_b}\sum_j^{N_b}\left(NN(x_j,t_j,s_j;\theta) - u_{FD}(x_j,t_j,s_j)\right)^2$$

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

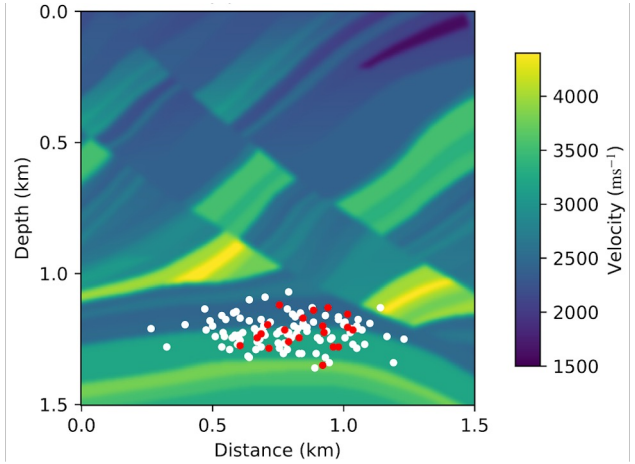
Randomly sampled input coordinates and source locations

Conditioned PINN

Conditioned PINNs

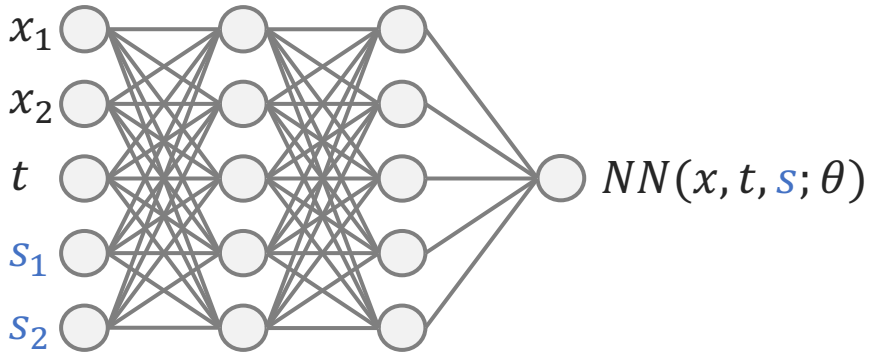


Velocity model, $c(x)$



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 diffusion-reaction equation:

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

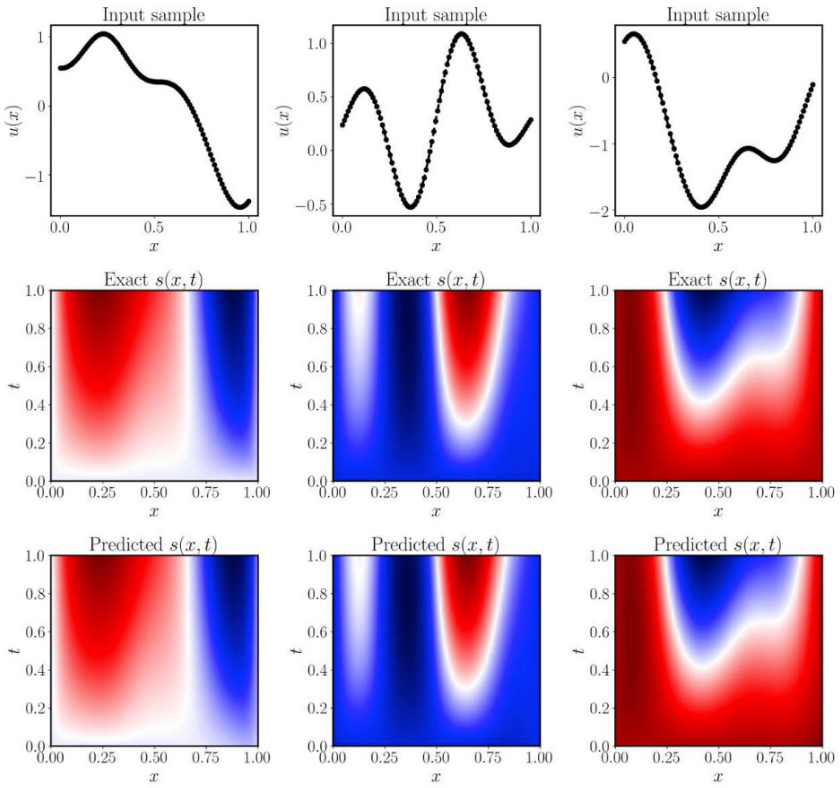
Input:
Discrete values of $u(x)$

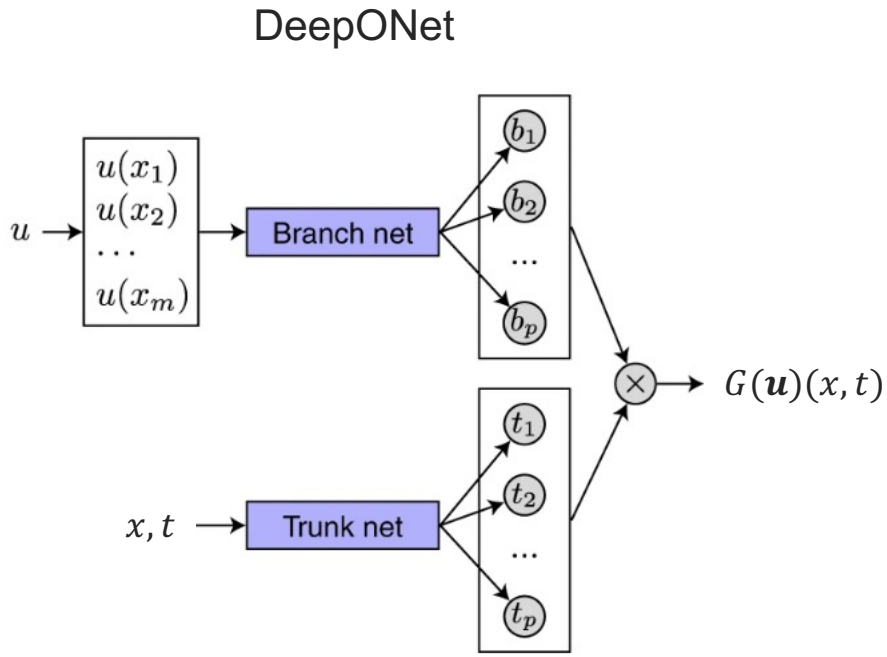
Output:

$$NN(x, t, \boldsymbol{u}; \theta) \approx s(x, t)$$

Architecture:
DeepONet

Trained using many examples of $u(x)$





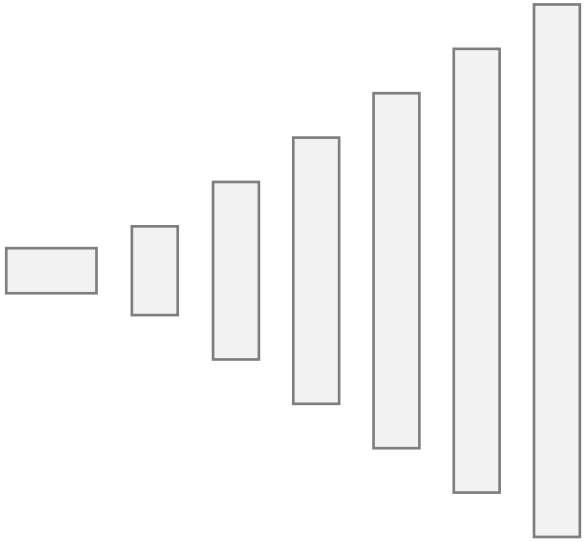
Wang et al, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Science Advances (2021)

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

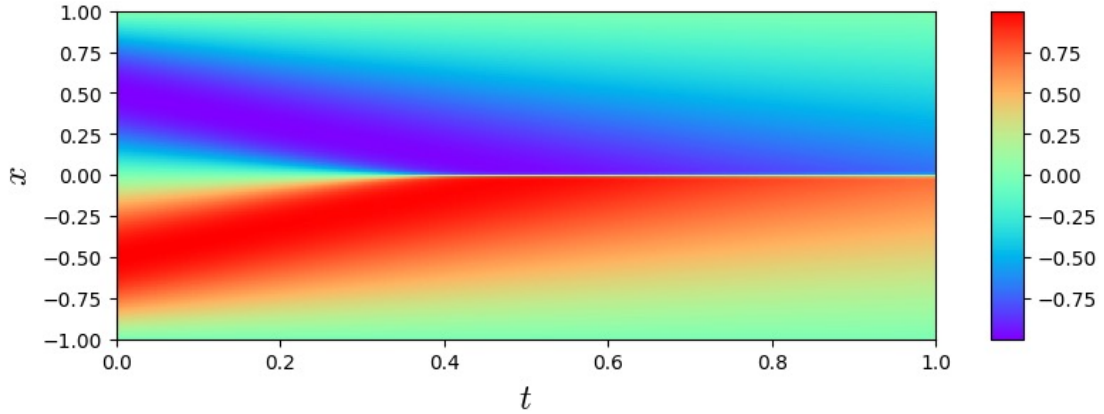
Discretised PINNs

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

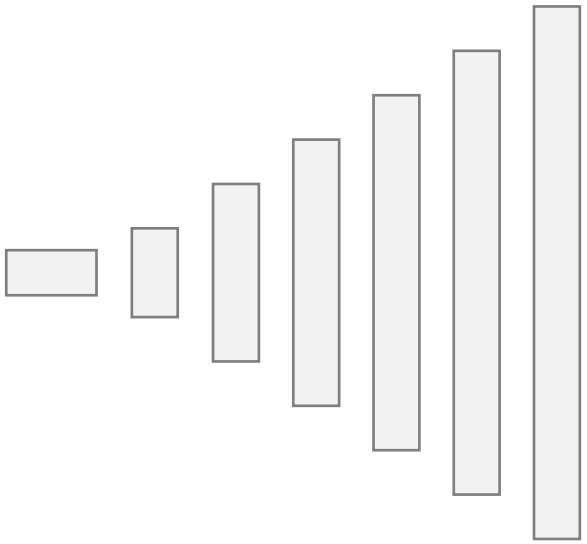
$$NN(\theta)_{ij} \approx u(x = x_i, t = t_j)$$

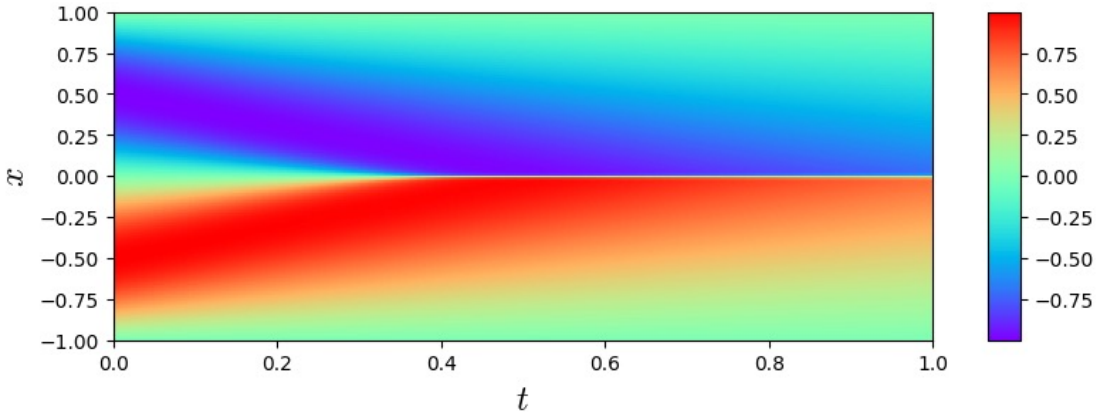
Instead of: $NN(x, t; \theta) \approx u(x, t)$

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

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

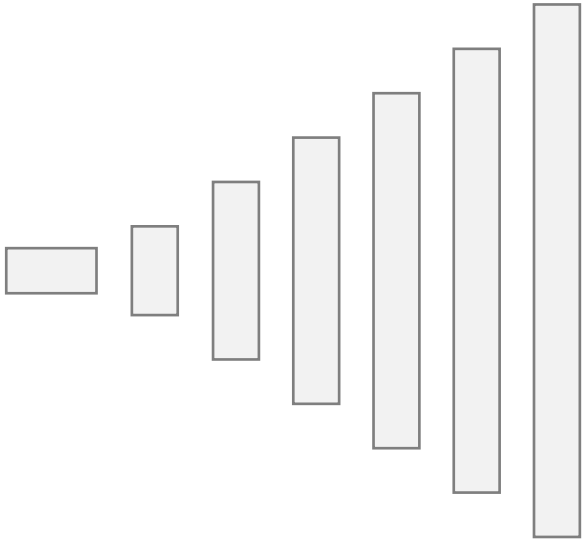
$$L_p(\theta) = \frac{1}{N_x N_t}\sum_i^{N_x}\sum_j^{N_t}\left(\frac{\delta NN(\theta)_{ij}}{\delta t} + NN(\theta)_{ij}\frac{\delta NN(\theta)_{ij}}{\delta x} - \nu\frac{\delta^2 NN(\theta)_{ij}}{\delta x^2}\right)^2$$

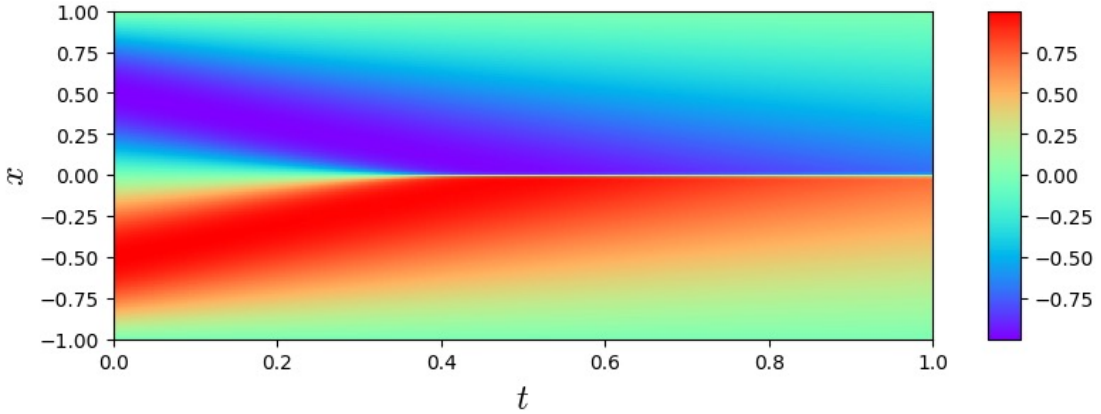
Discretised PINN

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.

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

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

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

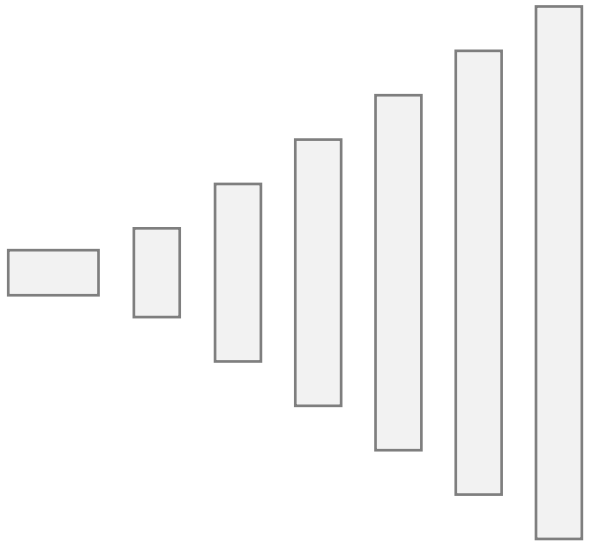
Discretised PINN

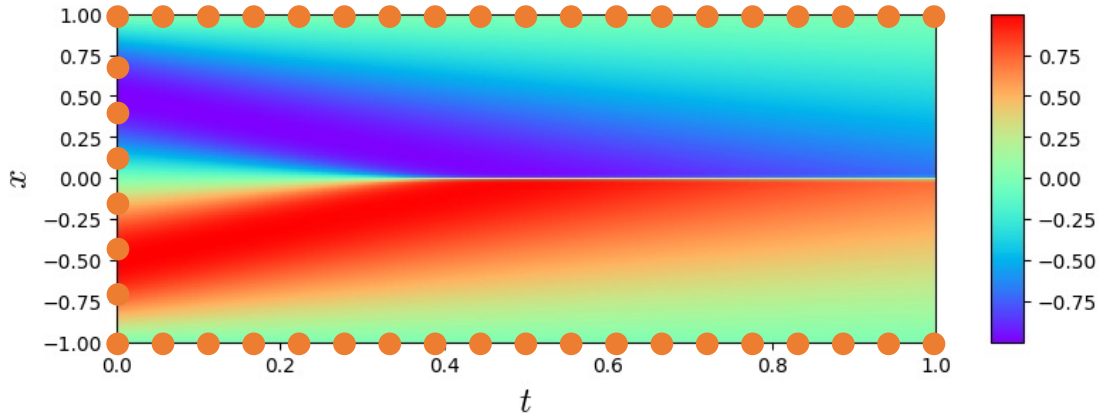
Autodiff is still used to update θ

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.

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

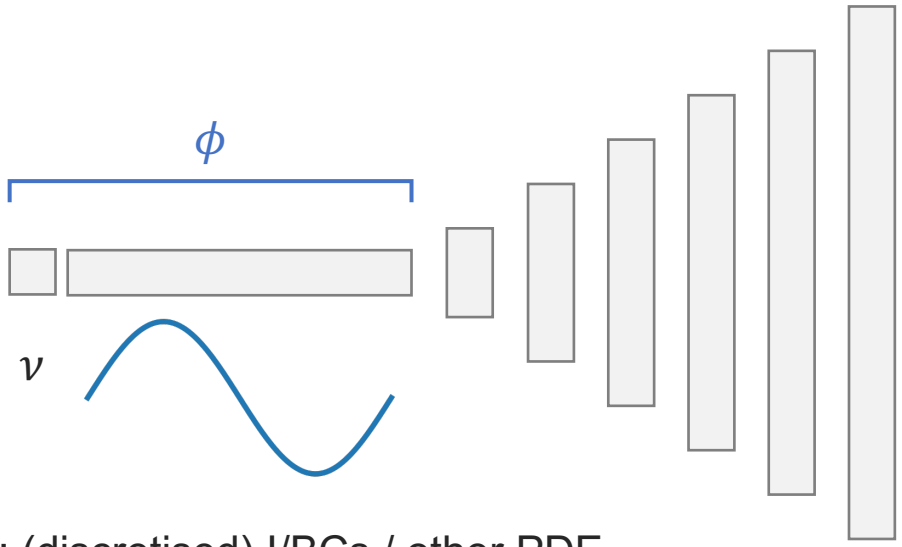
$$NN(\theta)_{ij} \approx u(x = x_i, t = t_j)$$

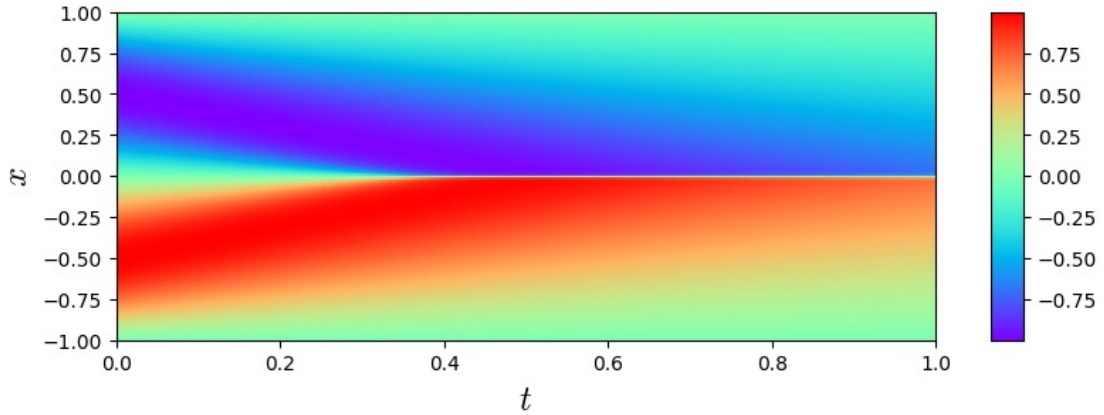
$$L_p(\theta) = \frac{1}{N_x N_t} \sum_i^{N_x} \sum_j^{N_t} \left(\frac{\delta NN(\theta)_{ij}}{\delta t} + NN(\theta)_{ij} \frac{\delta NN(\theta)_{ij}}{\delta x} - \nu \frac{\delta^2 NN(\theta)_{ij}}{\delta x^2} \right)^2$$

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

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

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

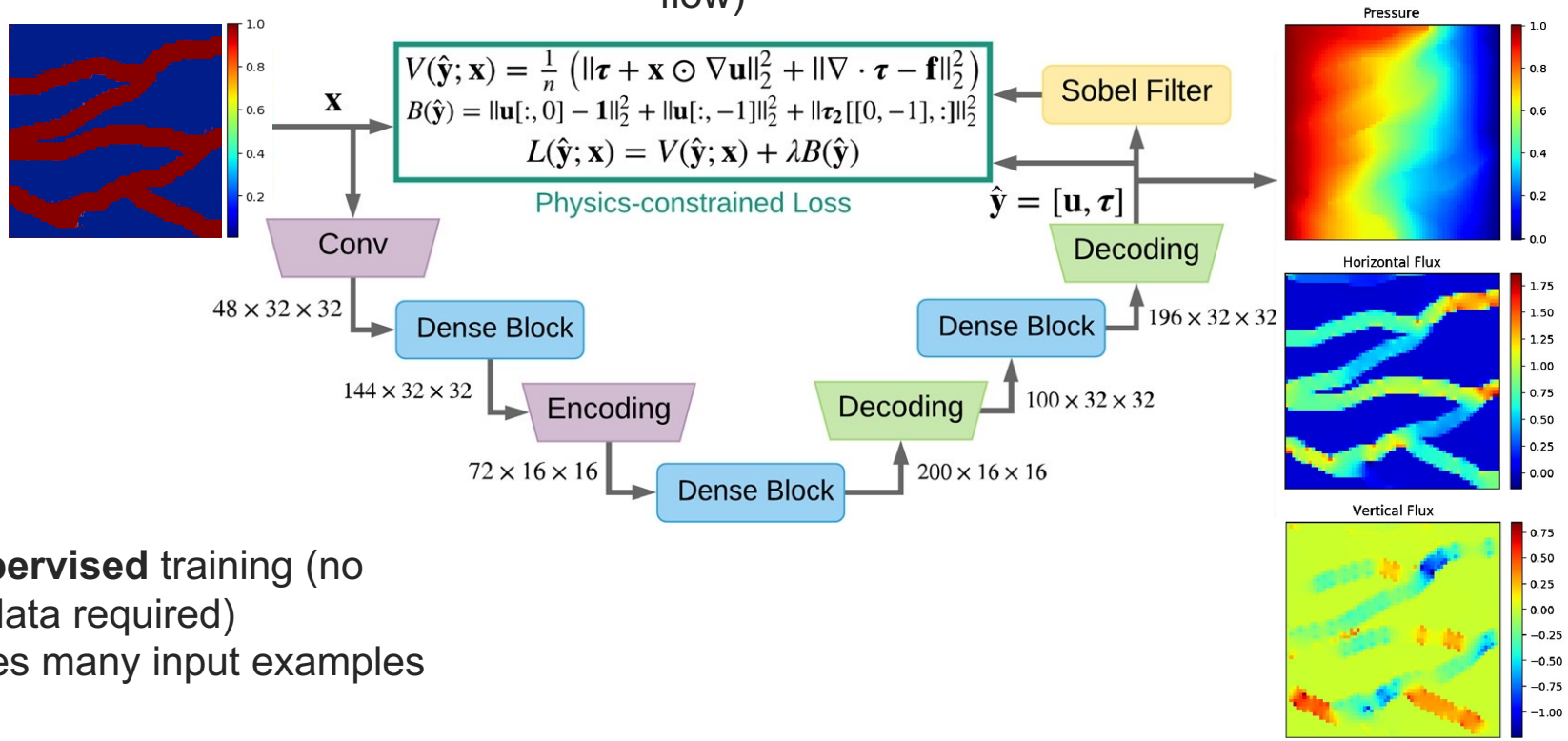
Conditioned discretised PINN

Conditioned discretised PINNs

Input: permeability field

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

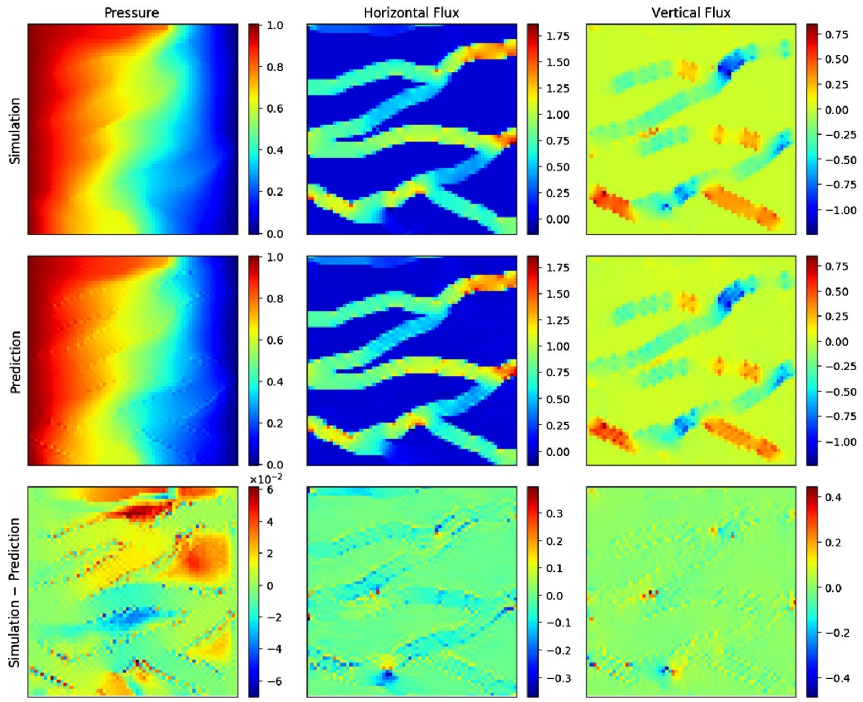
Output: fluid pressure and flux field



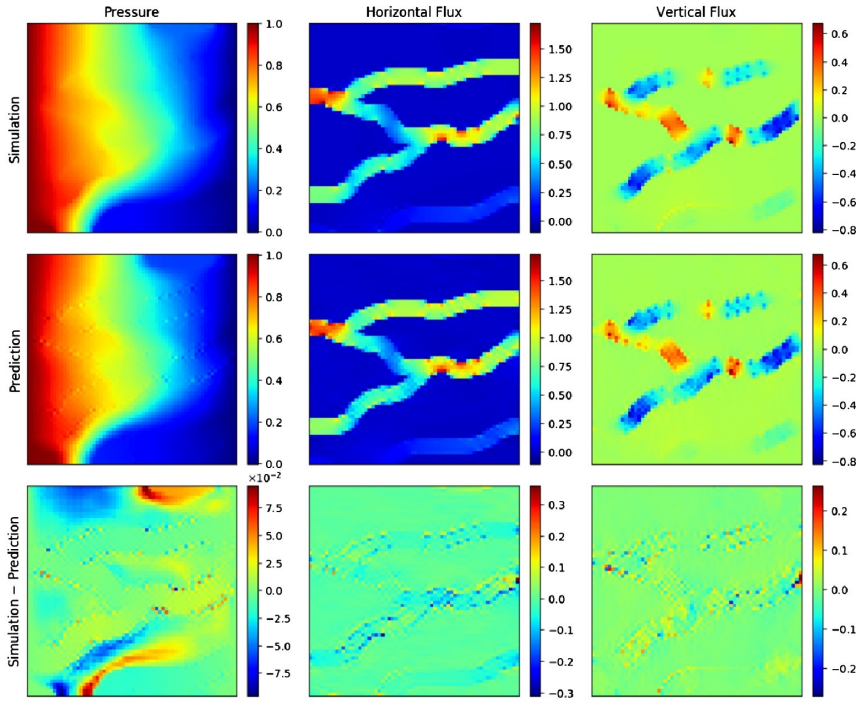
- Fully **unsupervised** training (no simulation data required)
- Only requires many input examples to train

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

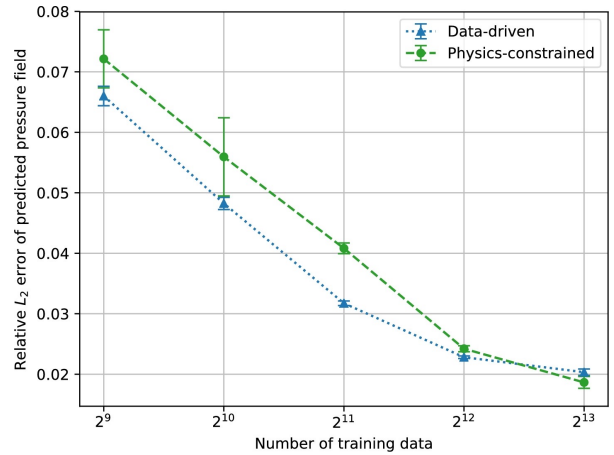


(c) Channelized, test 1.

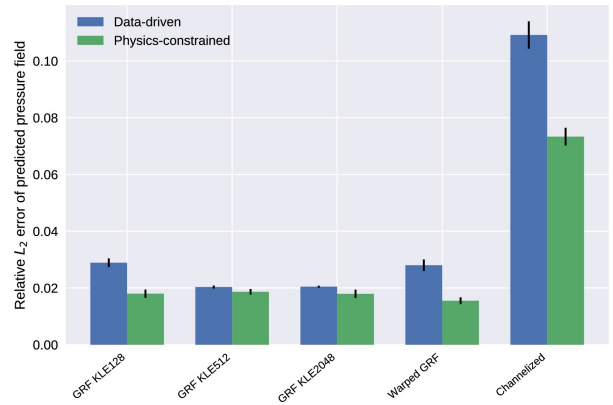


(d) Channelized, test 2.

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
- Fully unsupervised
- 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)

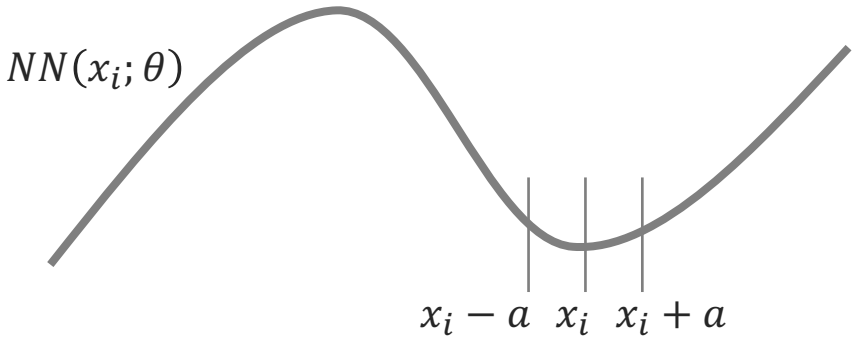
Gao et al, PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, JCP (2021)

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

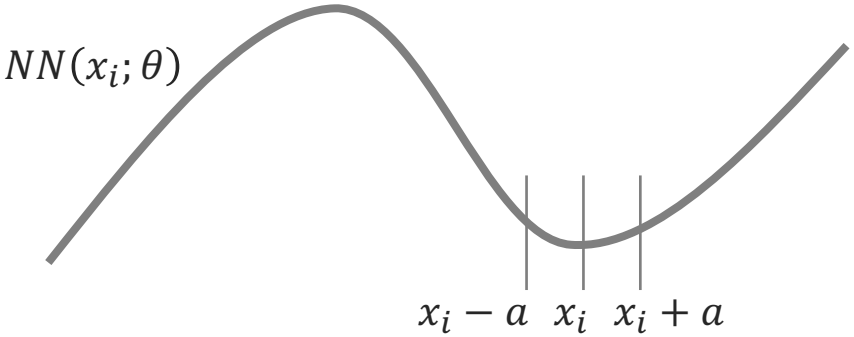
$$L_p(\theta) = \frac{1}{N_p}\sum_{i}^{N_p} \|\mathcal{D}[NN(x_i;\theta)] - f(x_i)\|^2$$

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



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

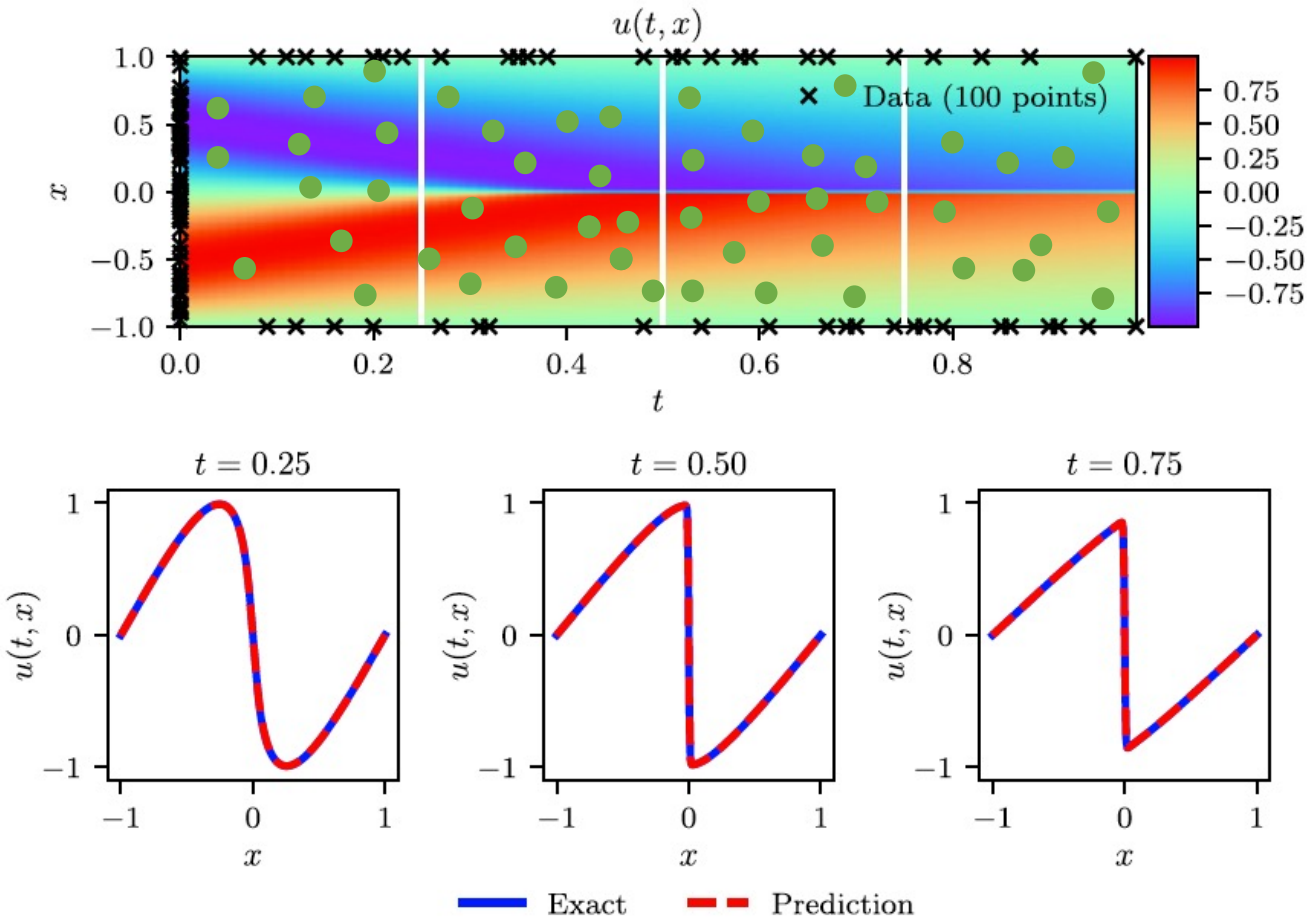
Can offer 2-4x speedups depending on PDE

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)

Limitation 2) – poor convergence

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)

$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_j^{N_{b1}} \left(NN(x_j, 0; \theta) + \sin(\pi x_j) \right)^2$$
$$+ \frac{\lambda_2}{N_{b2}} \sum_k^{N_{b2}} (NN(-1, t_k; \theta) - 0)^2$$
$$+ \frac{\lambda_3}{N_{b3}} \sum_l^{N_{b3}} (NN(+1, t_l; \theta) - 0)^2$$
$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \left(\left(\frac{\partial NN}{\partial t} + NN \frac{\partial NN}{\partial x} - \nu \frac{\partial^2 NN}{\partial x^2} \right) (x_i, t_i; \theta) \right)^2$$

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

Hard initial / boundary conditions

Idea: use neural network as part of a solution **ansatz**

Burgers' problem above:

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

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

Then let the solution approximated by

$$\begin{aligned}\hat{u}(x,t;\theta) &= (x-1)(x+1)(t-0)NN(x,t;\theta) - \sin(\pi x)\\ &\approx u(x,t)\end{aligned}$$

It is easy to verify that the initial / boundary conditions are now satisfied by **construction**

Lagaris et al, Artificial neural networks for solving ordinary and partial differential equations, IEEE (1998)

Hard initial / boundary conditions

Idea: use neural network as part of a solution **ansatz**

Burgers' problem above:

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

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

Then let the solution approximated by

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

It is easy to verify that the initial / boundary conditions are now satisfied by **construction**

Lagaris et al, Artificial neural networks for solving ordinary and partial differential equations, IEEE (1998)

Thus, the boundary loss $L_b(\theta)$ is not needed and the problem is turned into an **unconstrained** optimisation problem, where we minimize

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

Note:

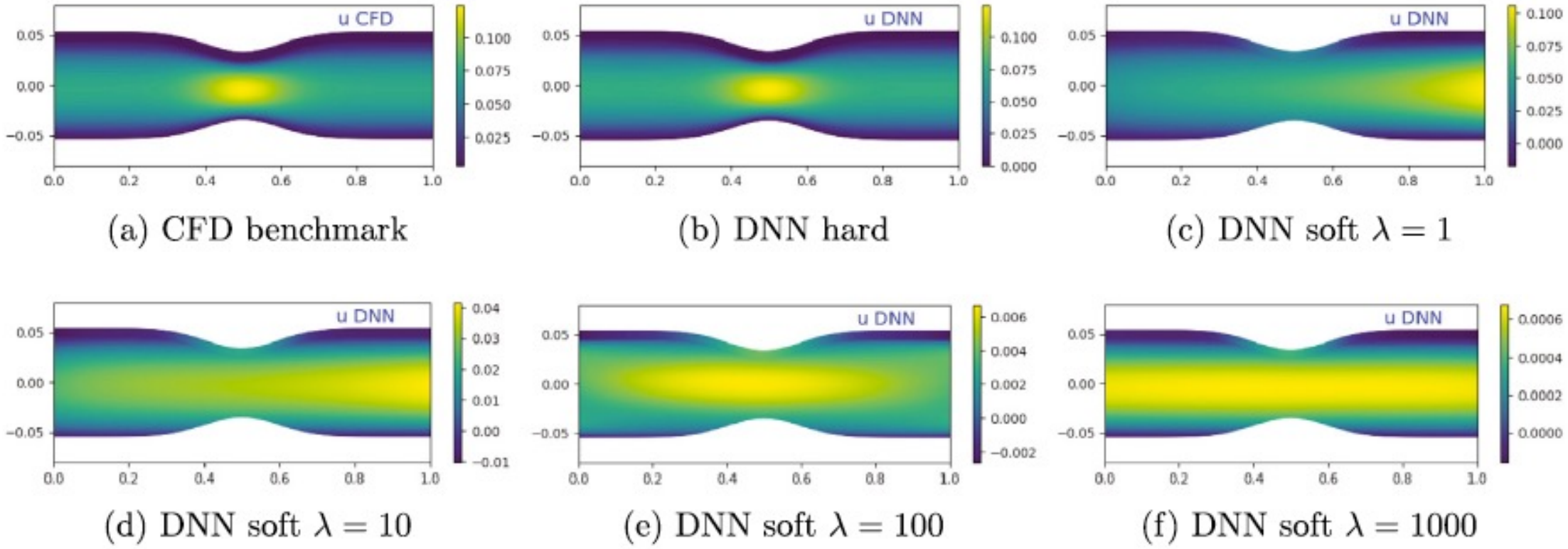
- You can think of the ansatz as a custom final layer of the network, asserting a **hard** constraint on the network's output

- Autodifferentiation lets us differentiate through ansatz

- Training is now fully **unsupervised** (only need collocation points)

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

Hard vs soft PINN



Sun et al, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Computer Methods in Applied Mechanics and Engineering (2020)

Soft PINN with varying λ vs hard PINN
When solving incompressible Navier-Stokes equations

Adaptive lambdas

Idea: Specify **separate** λ for each training point
And treat λ as **learnable**

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

$$L_b(\theta, \lambda) = \sum_k \sum_j \lambda_{kj} \left\| \mathcal{B}_k\left[NN(x_{kj}; \theta)\right] - g_k(x_{kj}) \right\|^2$$

$$L_p(\theta, \lambda) = \sum_i \lambda_i \left\| \mathcal{D}[NN(x_i; \theta)] - f(x_i) \right\|^2$$

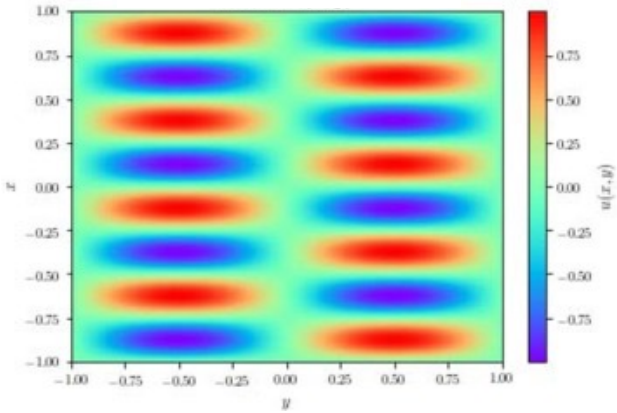
Loop:

1. Compute gradients $\frac{\partial L}{\partial \theta_l}, \frac{\partial L}{\partial \lambda_l}$
2. Update weights,
   $\theta_l \leftarrow \theta_l - \gamma \frac{\partial L}{\partial \theta_l}$ (minimise L)
3. Update lambdas,
   $\lambda_l \leftarrow \lambda_l + \gamma \frac{\partial L}{\partial \lambda_l}$ (**maximise** L)

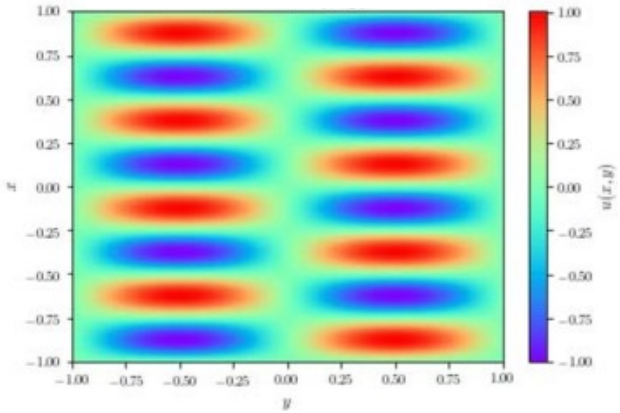
Note, for example, $\frac{\partial L}{\partial \lambda_i} = \left\| \mathcal{D}[NN(x_i; \theta)] - f(x_i) \right\|^2$

λ can be thought of as a soft attention layer

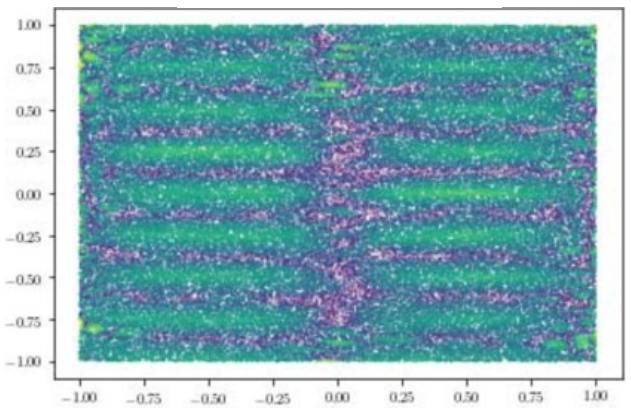
Helmholtz equation
Exact solution $u(x, y)$



PINN solution $NN(x, y; \theta)$

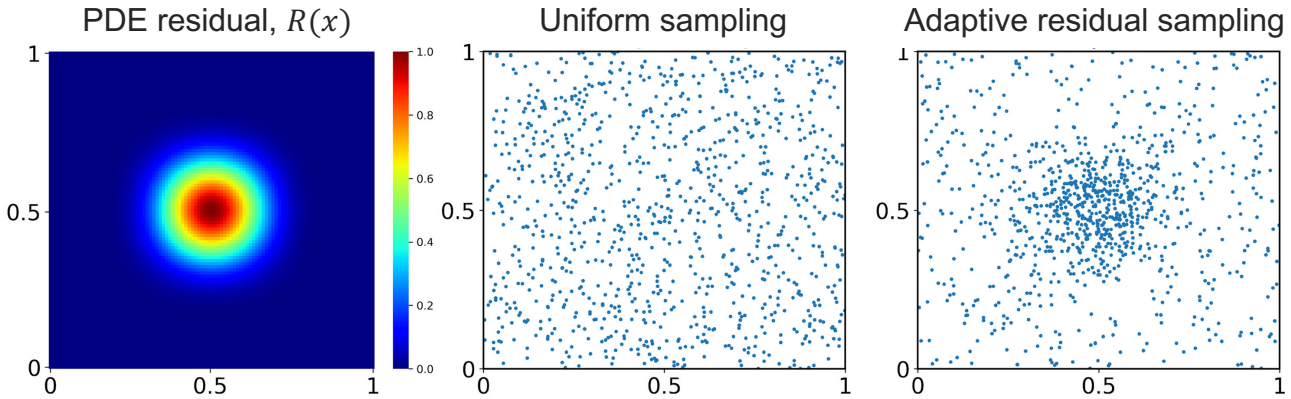


Final λ values
(brighter = larger)



McClenny et al, Self-adaptive physics-informed neural networks, JCP (2023)

Adaptive collocation points



Idea: sample collocation points according to the **probability distribution**

$$x \sim p(x) = \frac{R(x)}{A}$$

Where R is the PDE residual, $R(x) = \|\mathcal{D}[NN(x;\theta)] - f(x)\|^2$

And A is a normalising constant

Inspired by adaptive mesh refinement in FEM

Table 2. L^2 relative error of the PINN solution for the forward problems.

	Diffusion	Burgers'	Allen–Cahn	Wave
No. of residual points	30	2000	1000	2000
Grid	0.66±0.06%	13.7±2.37%	93.4±6.98%	81.3±13.7%
Random	0.74±0.17%	13.3±8.35%	22.2±16.9%	68.4±20.1%
LHS	0.48±0.24%	13.5±9.05%	26.6±15.8%	75.9±33.1%
Halton	0.24±0.17%	4.51±3.93%	0.29±0.14%	60.2±10.0%
Hammersley	0.17±0.07%	3.02±2.98%	**0.14±0.14%**	58.9±8.52%
Sobol	0.19±0.07%	3.38±3.21%	0.35±0.24%	57.5±14.7%
Random-R	**0.12±0.06%**	1.69±1.67%	0.55±0.34%	**0.72±0.90%**
RAR-G [3]	0.20±0.07%	**0.12±0.04%**	0.53±0.19%	0.81±0.11%
RAD	**0.11±0.07%**	**0.02±0.00%**	**0.08±0.06%**	**0.09±0.04%**
RAR-D	**0.14±0.11%**	**0.03±0.01%**	**0.09±0.03%**	**0.29±0.04%**

Wu et al, A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering (2023)

Adaptive collocation points

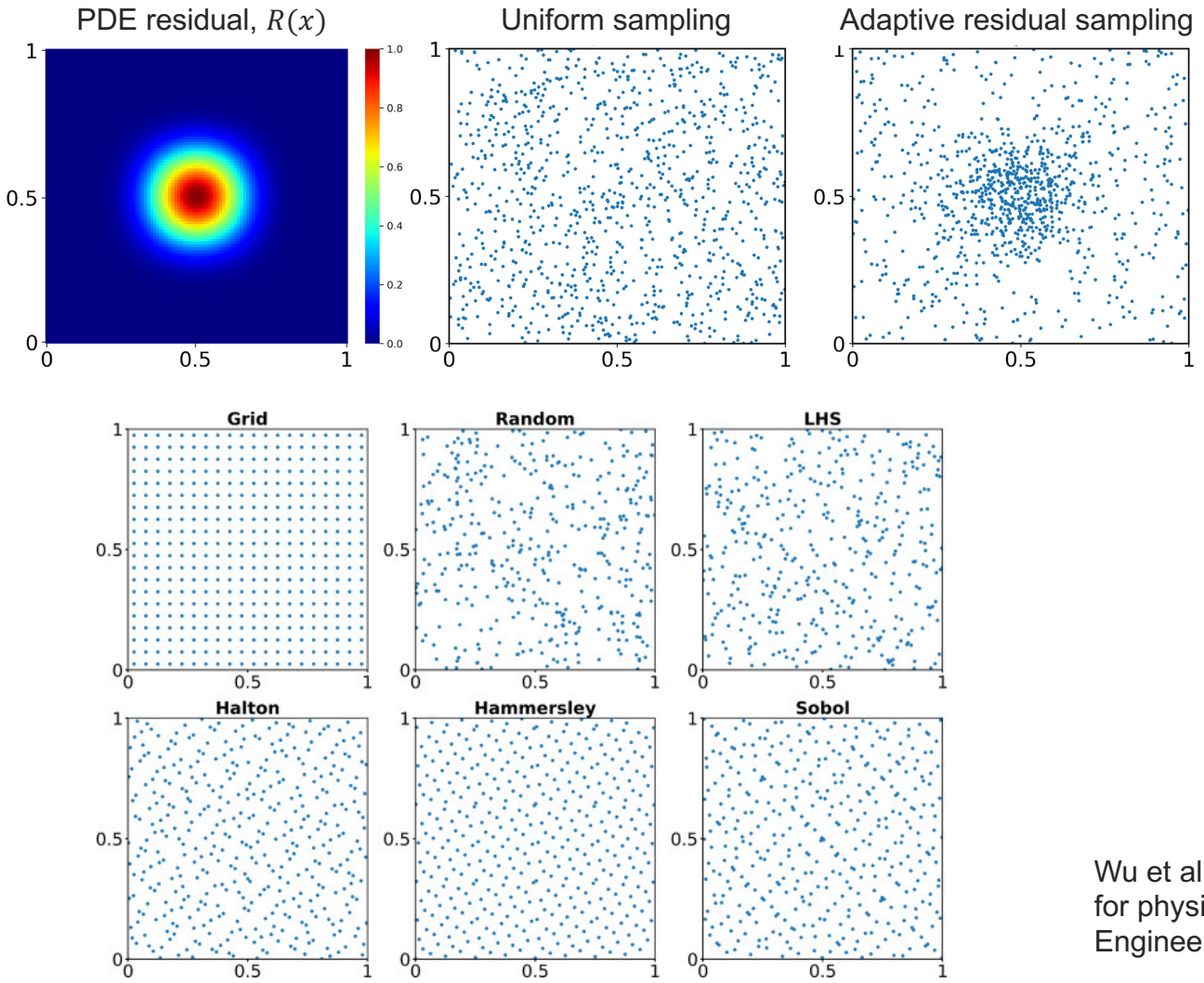


Table 2. L^2 relative error of the PINN solution for the forward problems.

	Diffusion	Burgers'	Allen–Cahn	Wave
No. of residual points	30	2000	1000	2000
Grid	0.66±0.06%	13.7±2.37%	93.4±6.98%	81.3±13.7%
Random	0.74±0.17%	13.3±8.35%	22.2±16.9%	68.4±20.1%
LHS	0.48±0.24%	13.5±9.05%	26.6±15.8%	75.9±33.1%
Halton	0.24±0.17%	4.51±3.93%	0.29±0.14%	60.2±10.0%
Hammersley	0.17±0.07%	3.02±2.98%	**0.14±0.14%**	58.9±8.52%
Sobol	0.19±0.07%	3.38±3.21%	0.35±0.24%	57.5±14.7%
Random-R	**0.12±0.06%**	1.69±1.67%	0.55±0.34%	**0.72±0.90%**
RAR-G [3]	0.20±0.07%	**0.12±0.04%**	0.53±0.19%	0.81±0.11%
RAD	**0.11±0.07%**	**0.02±0.00%**	**0.08±0.06%**	**0.09±0.04%**
RAR-D	**0.14±0.11%**	**0.03±0.01%**	**0.09±0.03%**	**0.29±0.04%**

Wu et al, A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering (2023)

5 min break

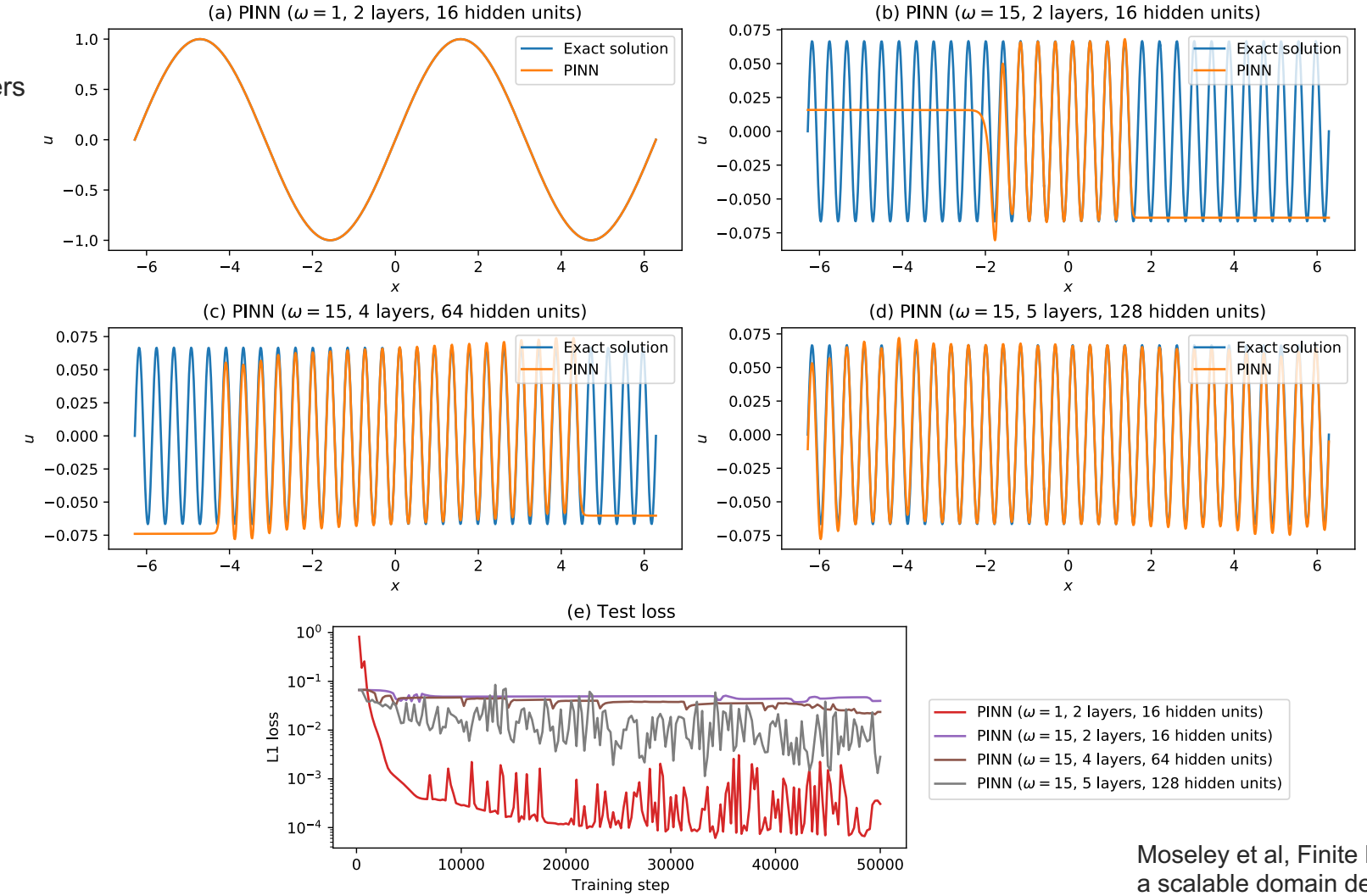
Lecture overview

- PINN applications: recap
- PINN limitations & extensions
  - Computational cost
  - Poor convergence
  - Scaling to more complex problems
- Summary: PINNs: when should I use them?

Limitation 3) – scaling to more complex problems

Scaling PINNs to higher frequencies

321 free parameters



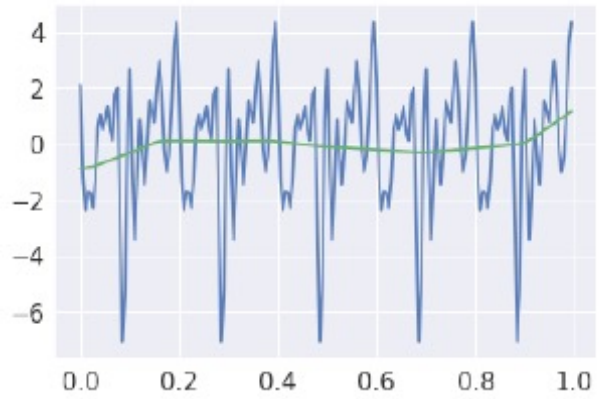
PINN solving:

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

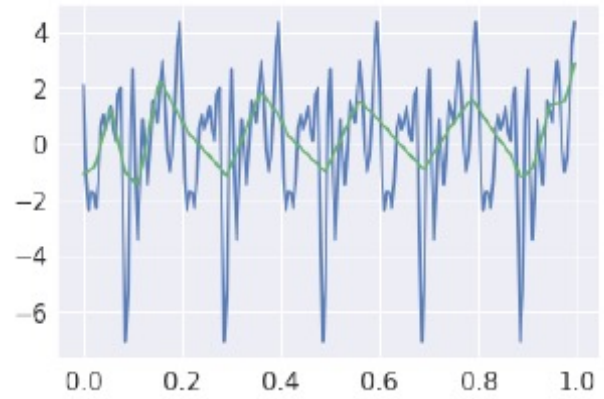
66,433 free parameters

Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ArXiv (2021)

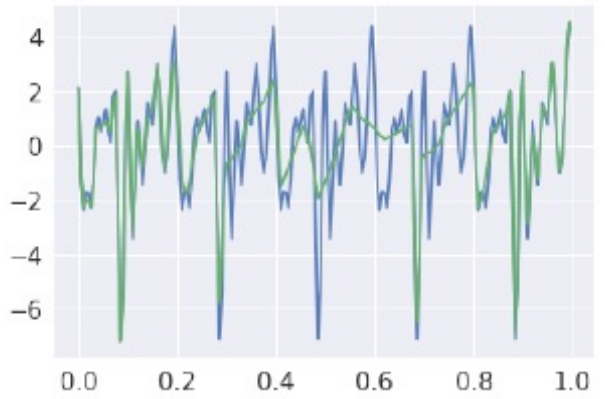
Scaling to higher frequencies: spectral bias issue



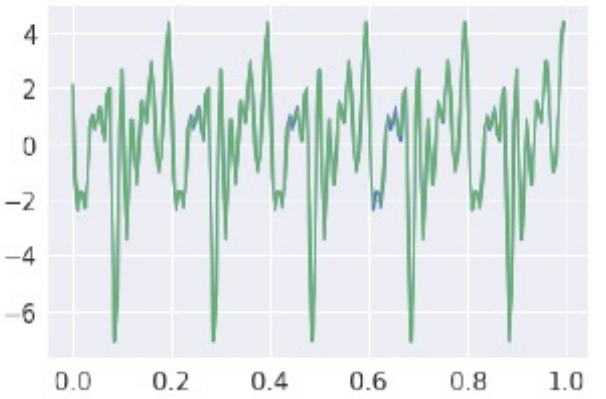
(a) Iteration 100



(b) Iteration 1000



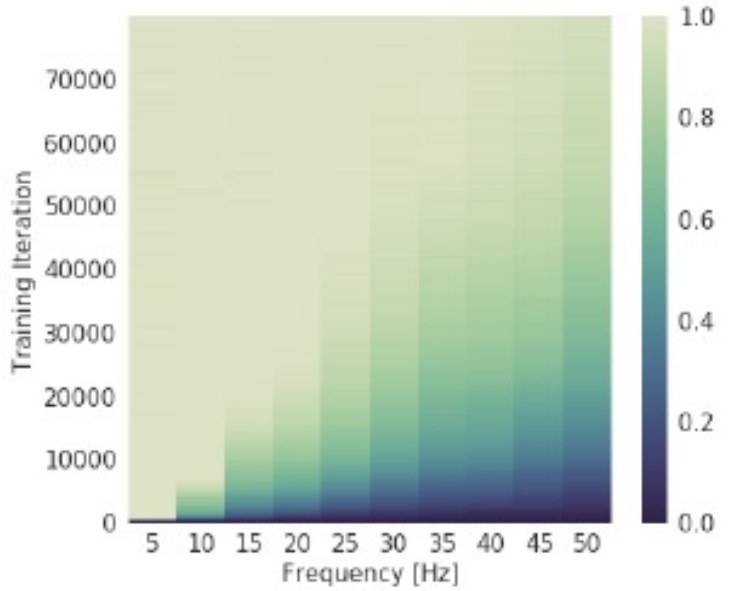
(c) Iteration 10000



(d) Iteration 80000

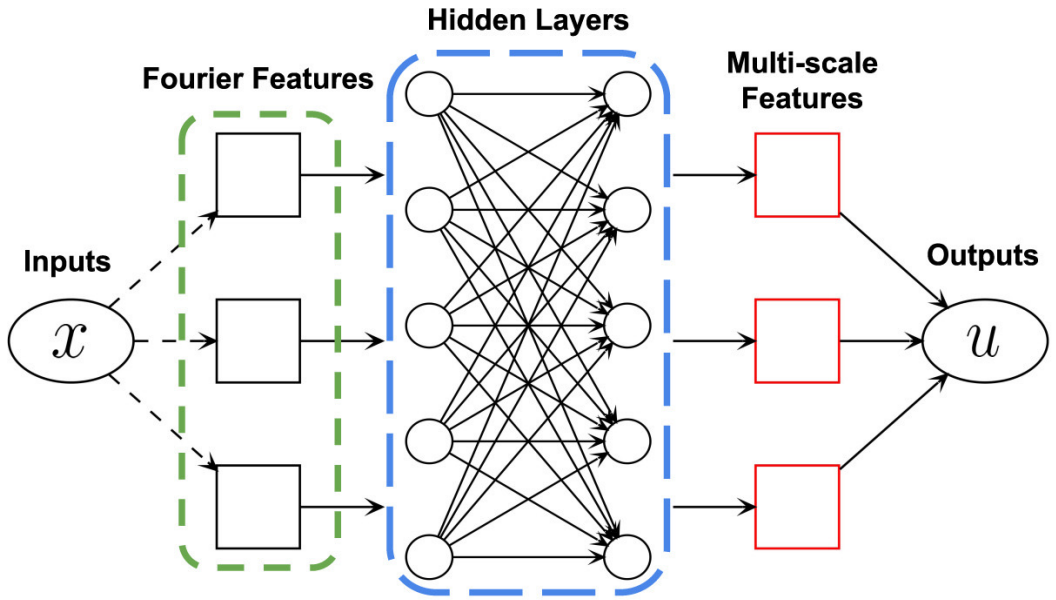
NNs prioritise learning **lower** frequency functions first

Under certain assumptions can be proved via neural tangent kernel theory



Rahaman, N., et al, On the spectral bias of neural networks. 36th International Conference on Machine Learning, ICML (2019)

Fourier features



Idea: **transform** input coordinates to higher-frequency functions

By using Fourier features:

$$\boldsymbol{\gamma}(\boldsymbol{x}) = [\cos(2\pi\Gamma\boldsymbol{x}), \sin(2\pi\Gamma\boldsymbol{x})]$$

Where Γ is a matrix of shape $(k \times d)$ where k is the number of Fourier features and d is the dimensionality of $\boldsymbol{x}$

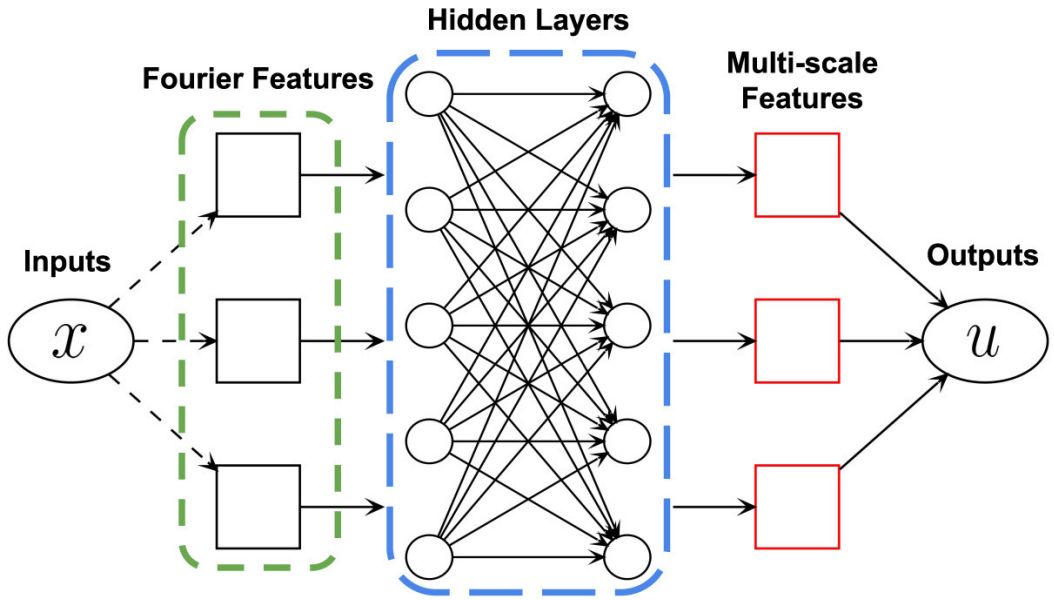
Typically, values of Γ are drawn from a normal distribution, i.e. $\Gamma_{ij} \sim \mathcal{N}(\cdot\,; \mu, \sigma)$, and fixed

μ, σ are hyperparameters which control the frequency of the Fourier features

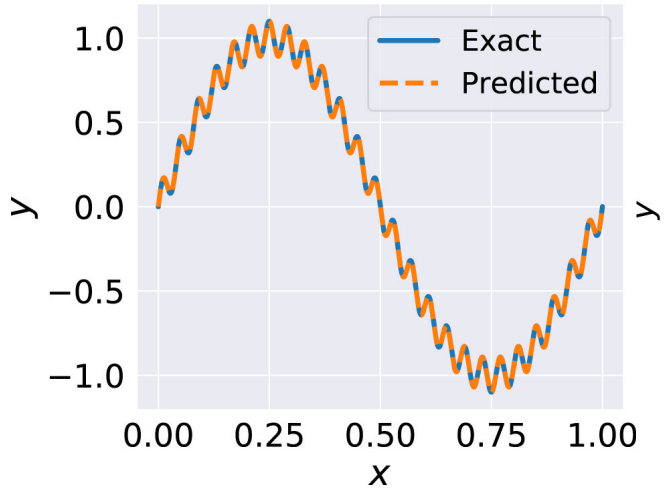
Tancik et al, Fourier features let networks learn high frequency functions in low dimensional domains, NeurIPS (2020)

Wang et al, On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering (2021)

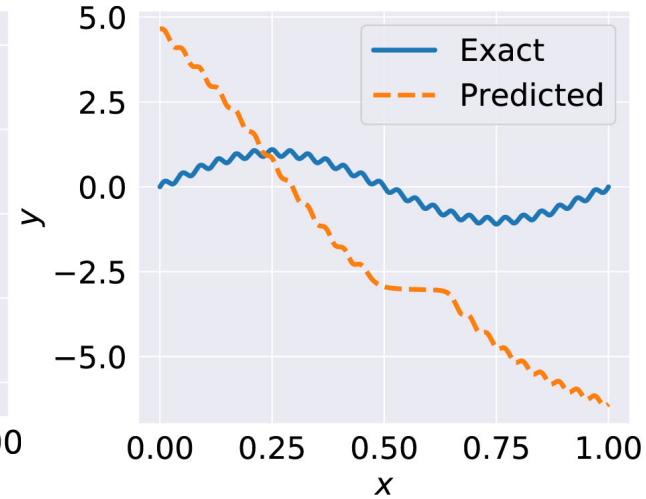
Fourier features



PINN with 2 sets of Fourier features



PINN



Solving 1D Poisson's equation

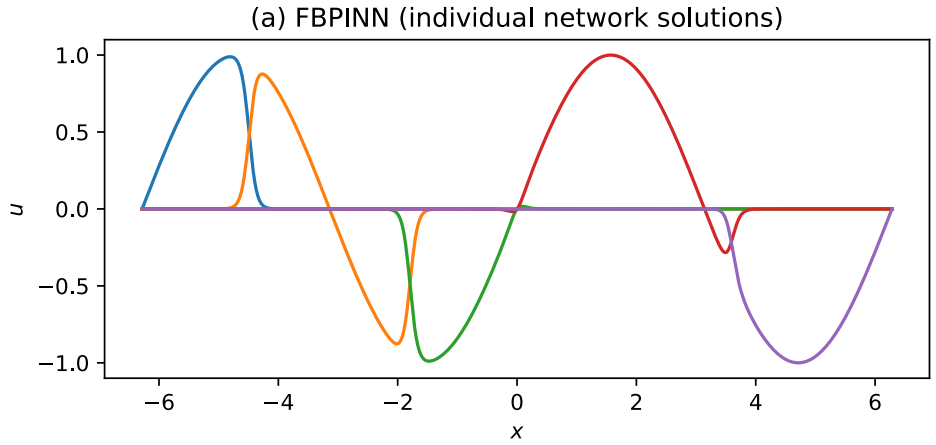
$\mu = 0$
$\sigma_1 = 1, \sigma_2 = 10$
$k = 25$

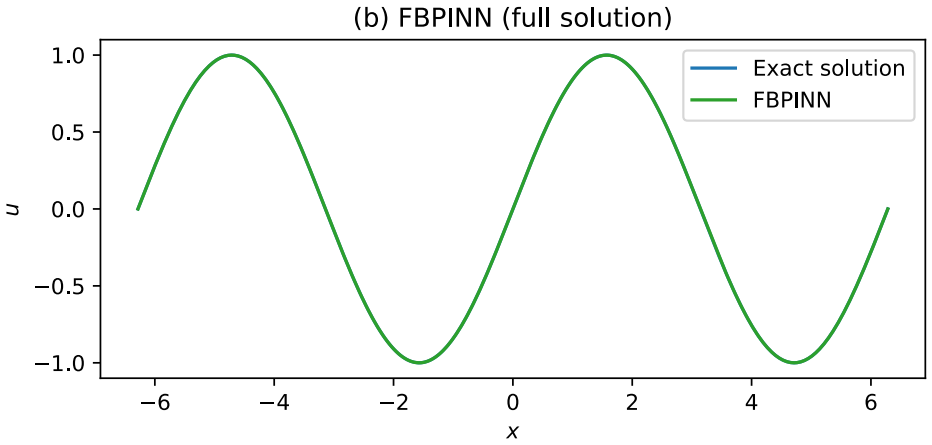
Fully connected network with 2 layers, 100 hidden units
Tanh activation function
Adam optimiser

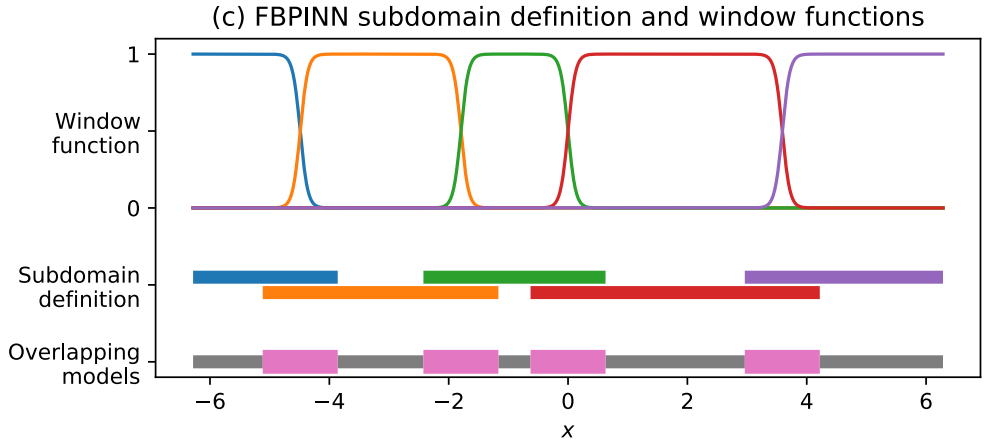
Tancik et al, Fourier features let networks learn high frequency functions in low dimensional domains, NeurIPS (2020)

Wang et al, On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering (2021)

Domain decomposition



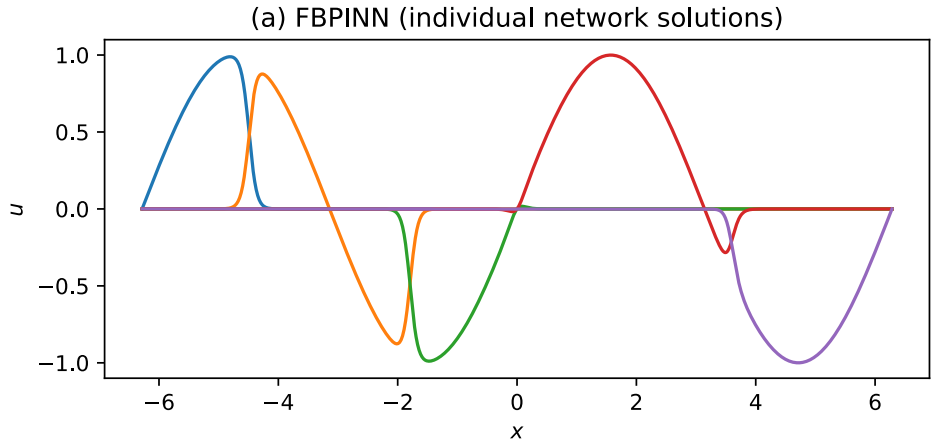


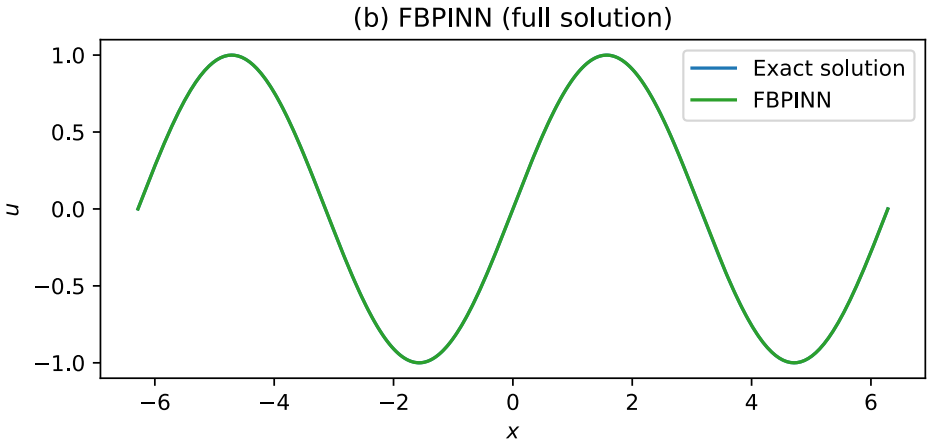


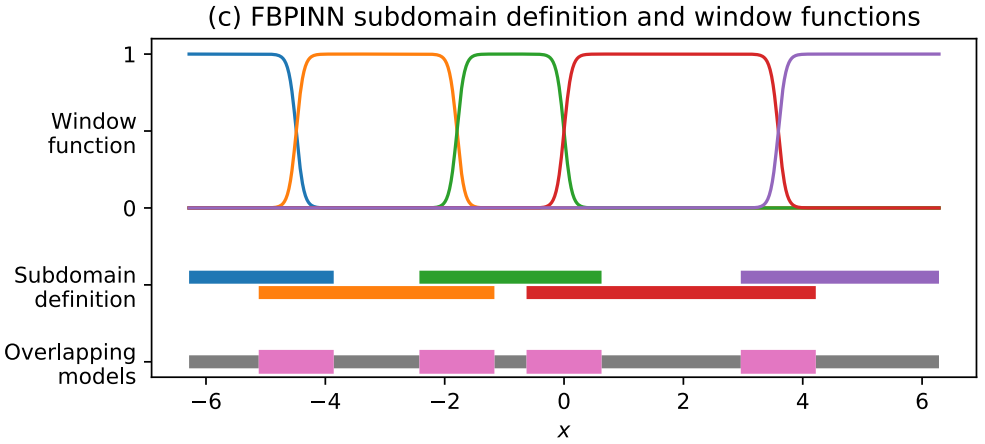
Idea: use a **divide-and-conquer** strategy to solve complex problems

Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ArXiv (2021)

Domain decomposition







Idea: use a **divide-and-conquer** strategy to solve complex problems

$$\hat{u}(x;\theta) = \mathcal{C}\left[\sum_i^n w_i(x) \cdot \text{unnorm} \circ NN_i \circ \text{norm}_i(x)\right]$$

Window function

Subdomain network

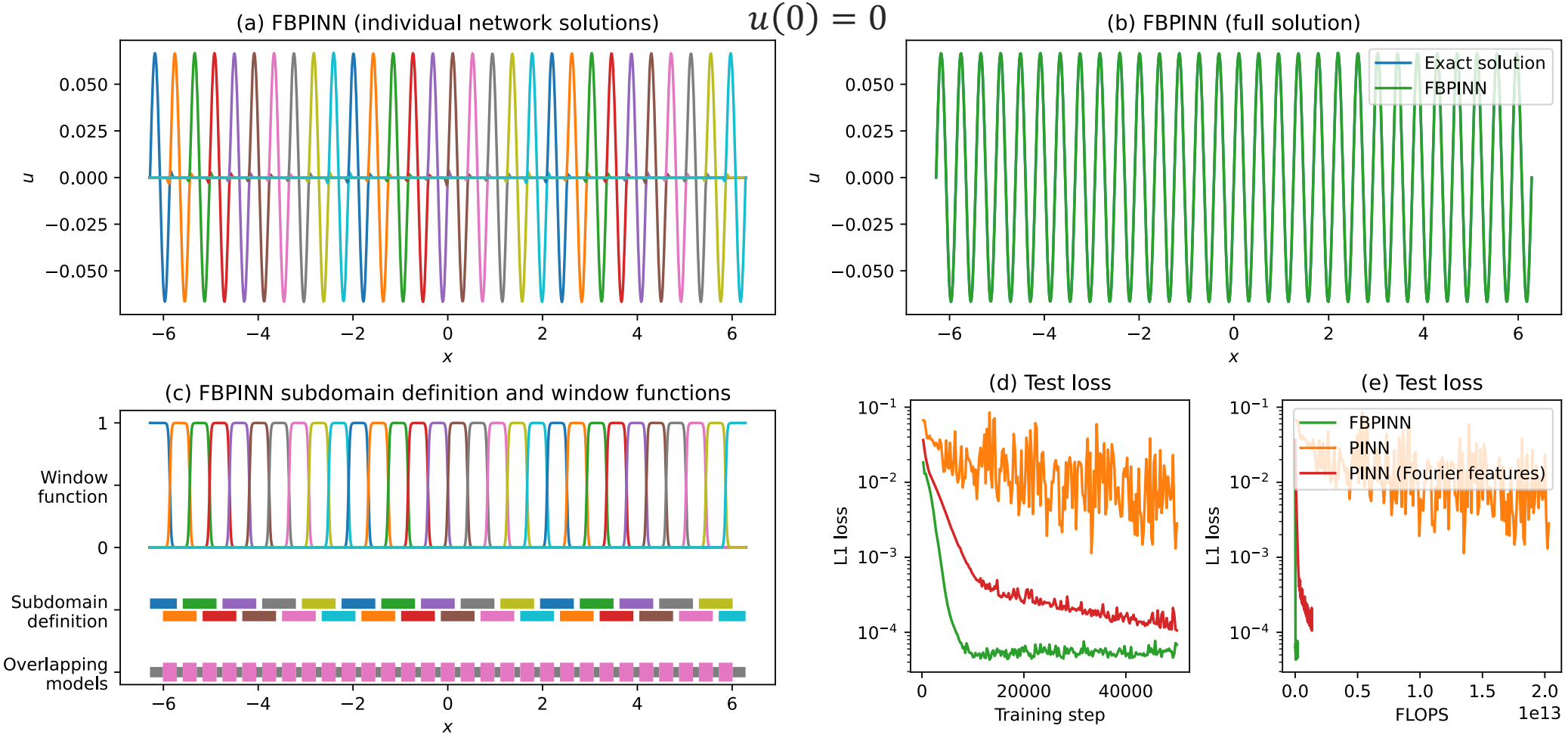
Individual subdomain normalisation

Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ArXiv (2021)

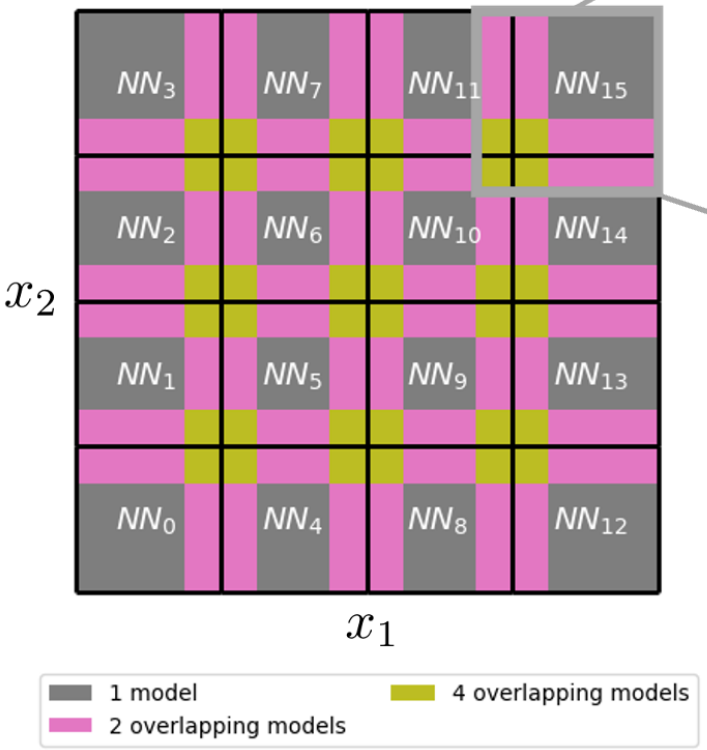
Domain decomposition

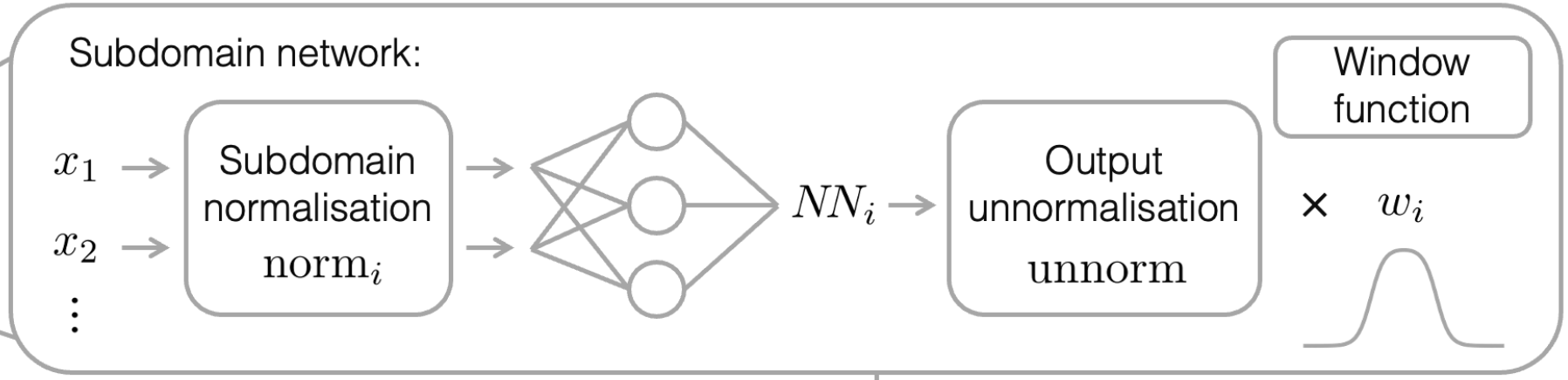
$$\frac{du}{dx} = \cos(\omega x)$$
$$u(0) = 0$$

Subdomain network size: 2 layers, 16 hidden units



Extension to multiple dimensions





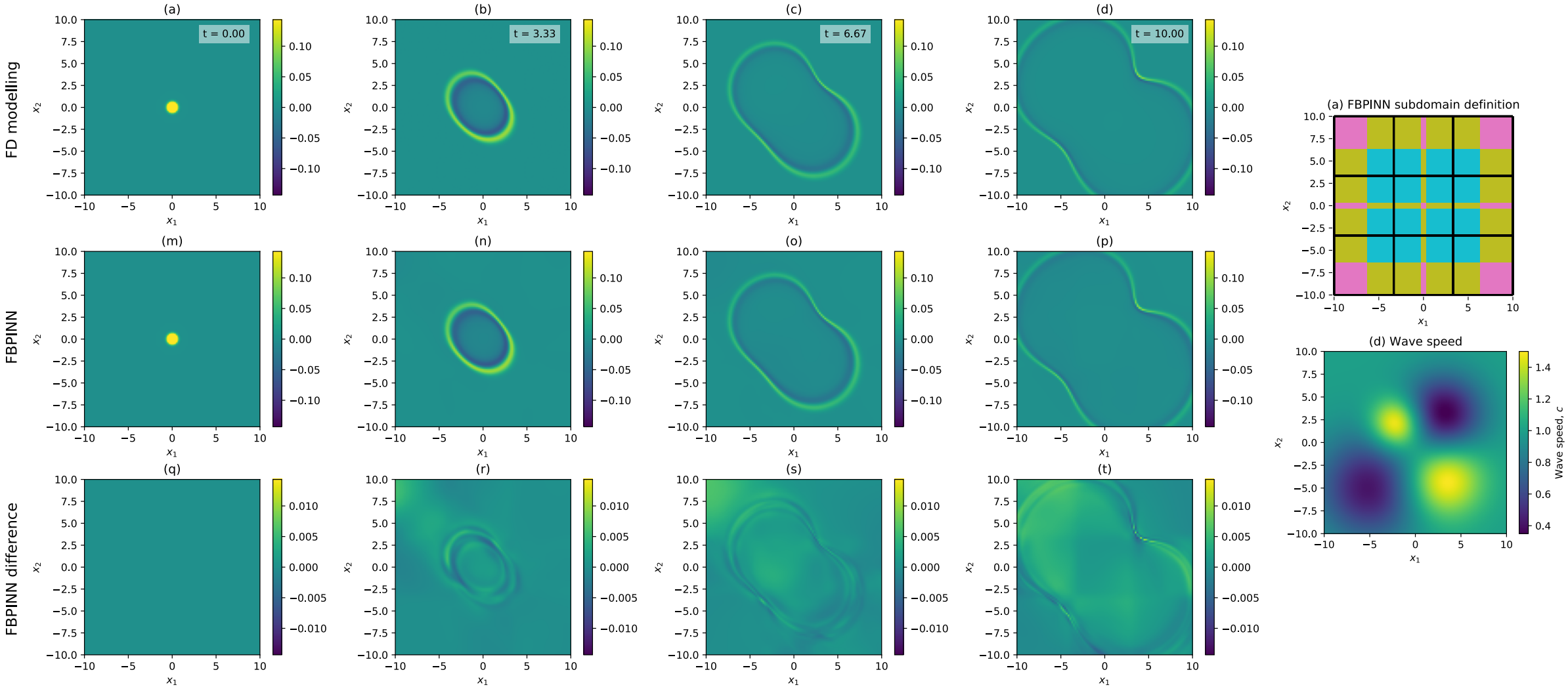
FBPINN solution:

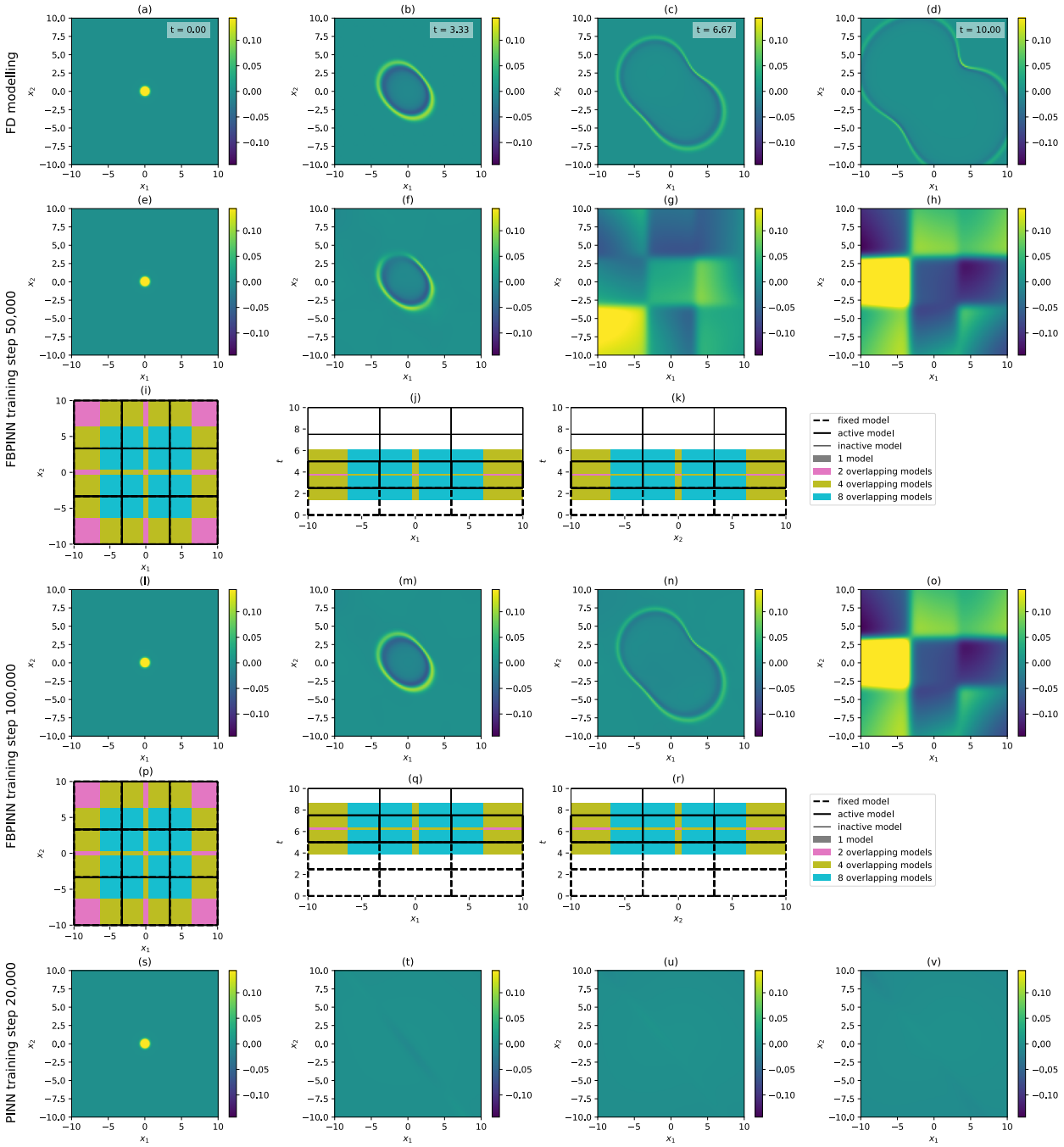
$$\hat{u}(x;\theta) = \mathcal{C}\left[\sum_i^n w_i(x) \cdot \text{unnorm} \circ NN_i \circ \text{norm}_i(x)\right]$$

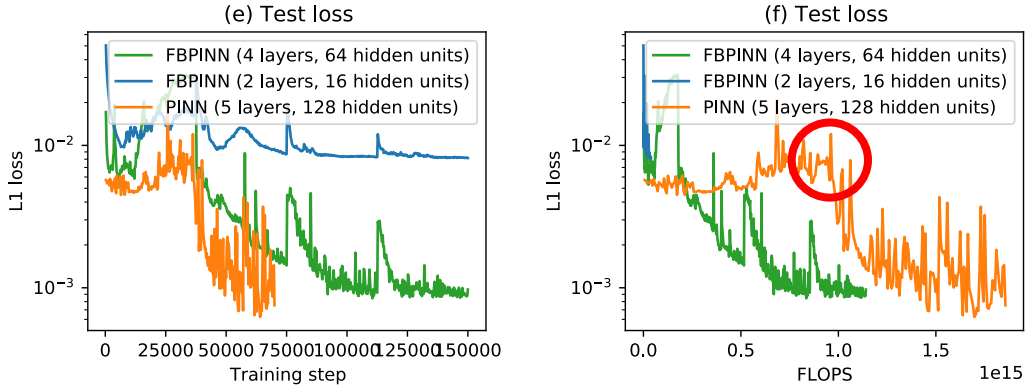
Summation over all subdomain networks

- In general, any overlapping domain decomposition could be used

FBPINNs for solving the wave equation







- FBPINN with time-stepping converges in a stable fashion
- Whilst PINN solution “thrashes”

Other PINN extensions

- Different differential equations:
  - Fractional equations, stochastic PDEs, variational formulations (VPINNs), mixed formulations, …
- Uncertainty estimation:
  - Bayesian PINNs (B-PINNs), GAN PINNs, …
- Architectures:
  - Adaptive activation functions, extreme learning machines, SIRENs, LSTMs, …
- PINN software libraries:
  - NVIDIA Modulus, DeepXDE, NeuroDiffEq, PyDENs, …
- And much more!

Advantages / limitations of PINNs

Advantages

- **Mesh-free**
- Can perform well for high-dimensional PDEs
- Can be **extended** to solve inverse and discovery problems
- Often perform best on “messy/**mixed**” problems, where some noisy data is available, physics is perhaps not perfectly known, and traditional algorithms require many forward evaluations
- Tractable, analytical solution gradients (e.g. for sensitivity analysis)
- Mostly **unsupervised**

Limitations

- **Computational cost** often high, especially for forward-only problems (requires retraining each time)
- Not guaranteed to converge / **convergence properties** less well understood
- Challenging to **scale** to more complex problems (larger domains, multi-scale, multi-physics)
- Often not obvious what appropriate neural network architecture is

- But.. many PINN extensions exist!

Lecture summary

- Whilst PINNs offer a powerful approach for solving problems related to differential equations, “vanilla” PINNs suffer from some major limitations
  - Computational cost
  - Poor convergence
  - Scaling to more complex problems
- There exist many (100-1000s) of PINN extensions and applications (only some of which were covered above)
- PINNs remain an active and fast-moving field of research!