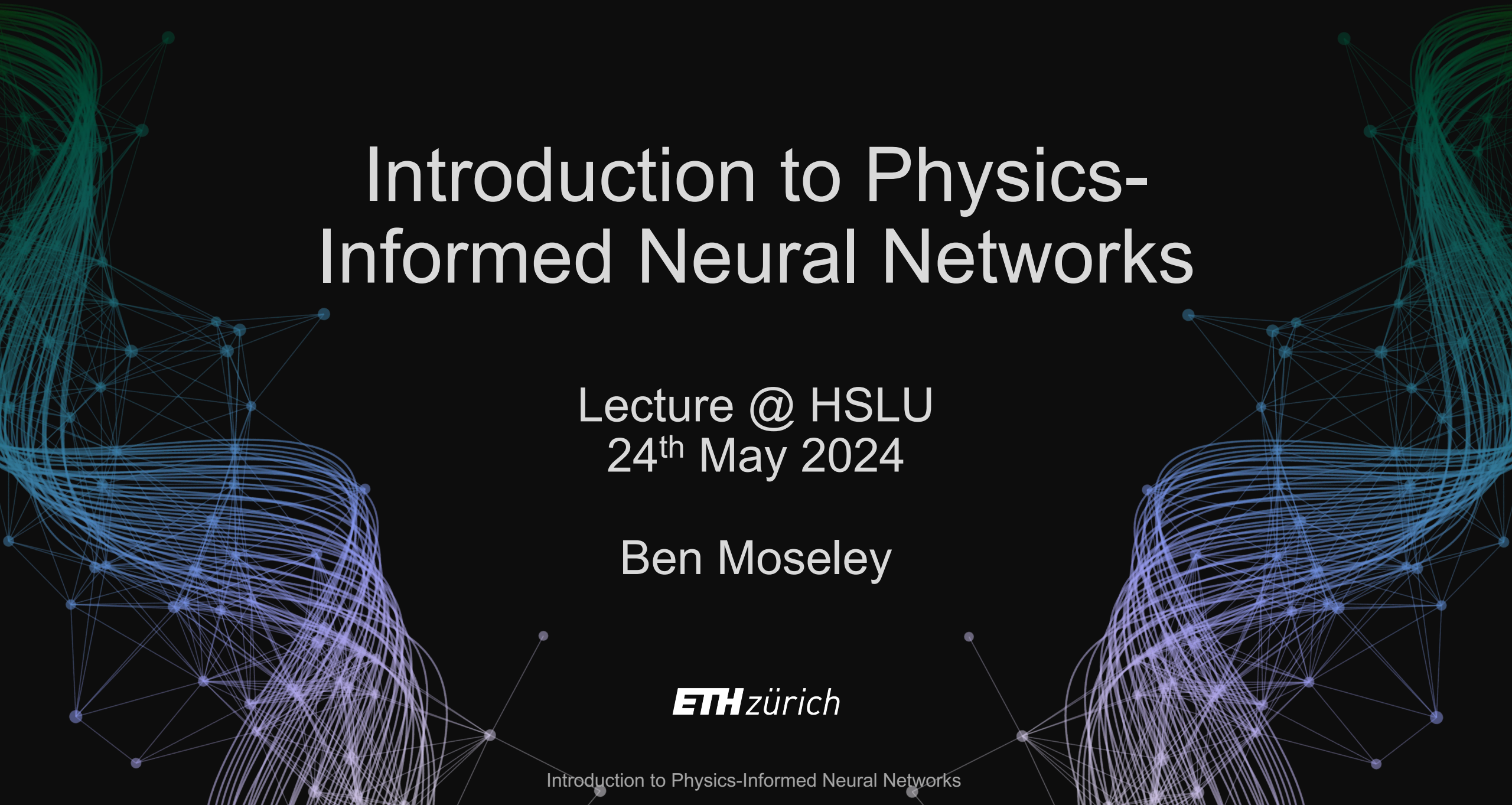


Introduction to Physics-Informed Neural Networks

Lecture @ HSLU
24th May 2024

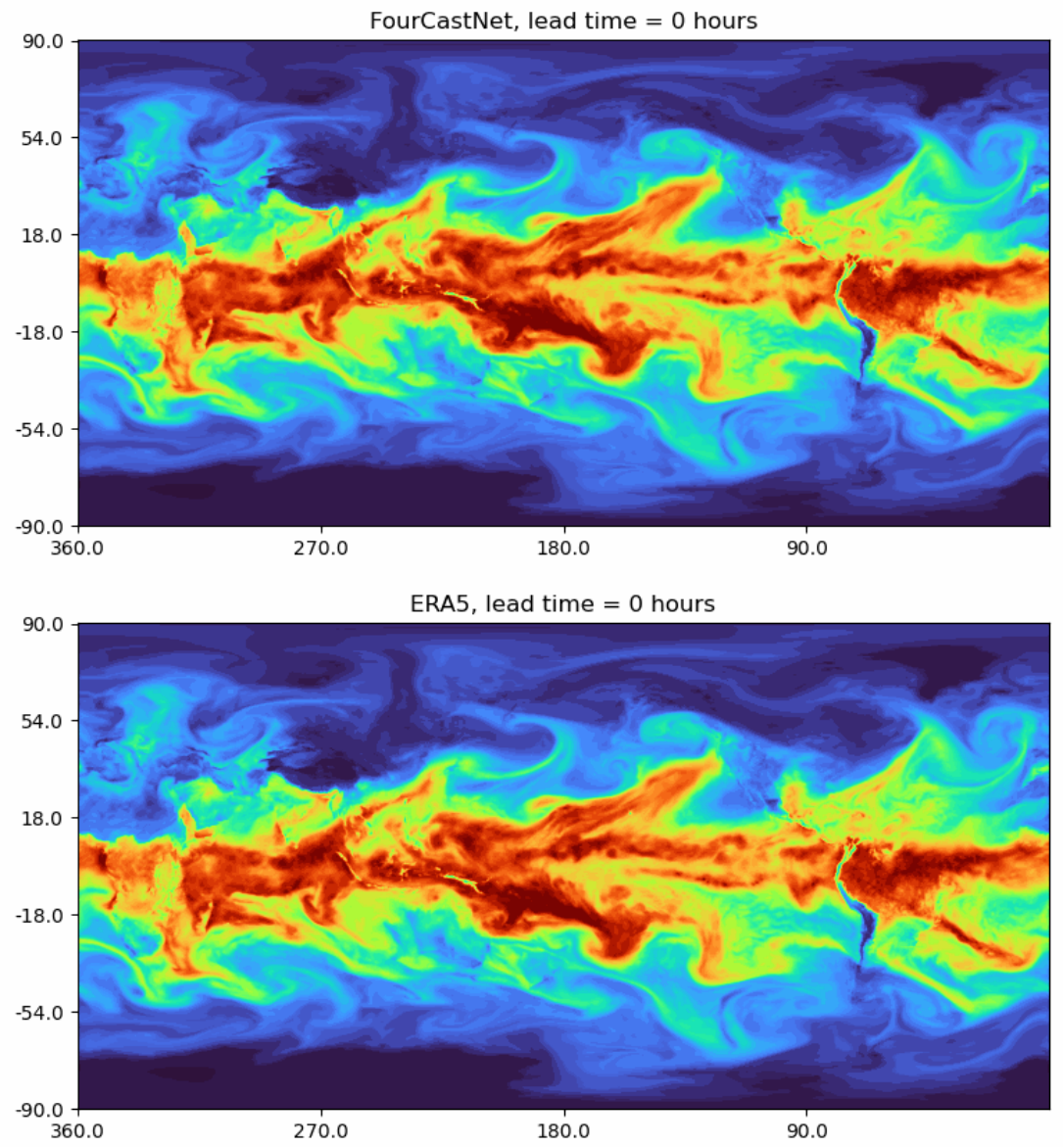
Ben Moseley

ETH zürich

Lecture overview

- Introduction to scientific machine learning (SciML)
- Rapid recap of deep learning fundamentals
- Introduction to physics-informed neural networks (PINNs)
- Live coding a PINN
- Current applications and limitations of PINNs

AI for science: a revolution?

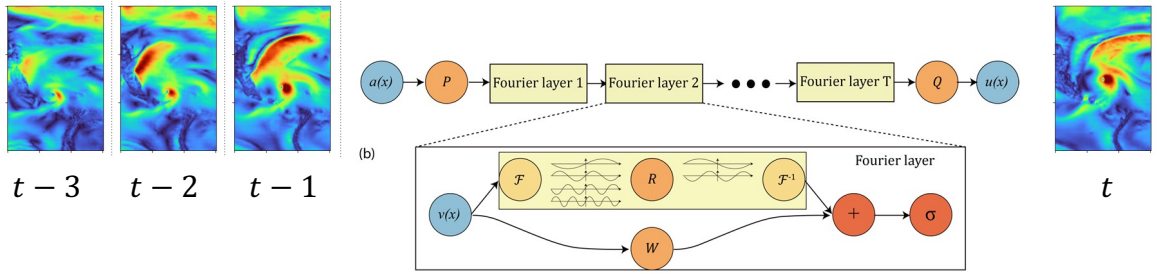


The Washington Post
Democracy Dies in Darkness

WEATHER Extreme Weather Climate Capital Weather Gang Environment Climate Lab

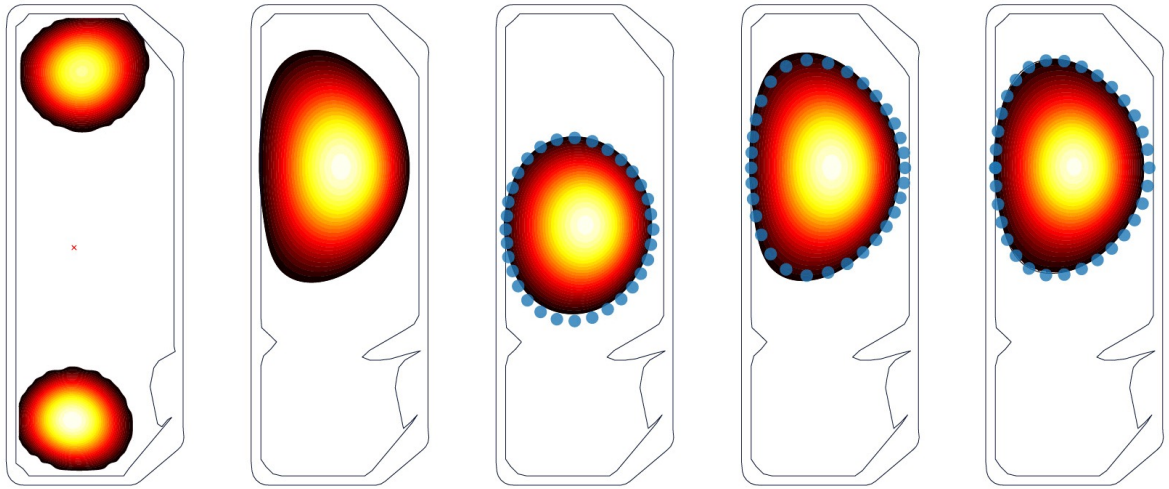
How Big Tech AI models nailed forecast for Hurricane Lee a week in advance

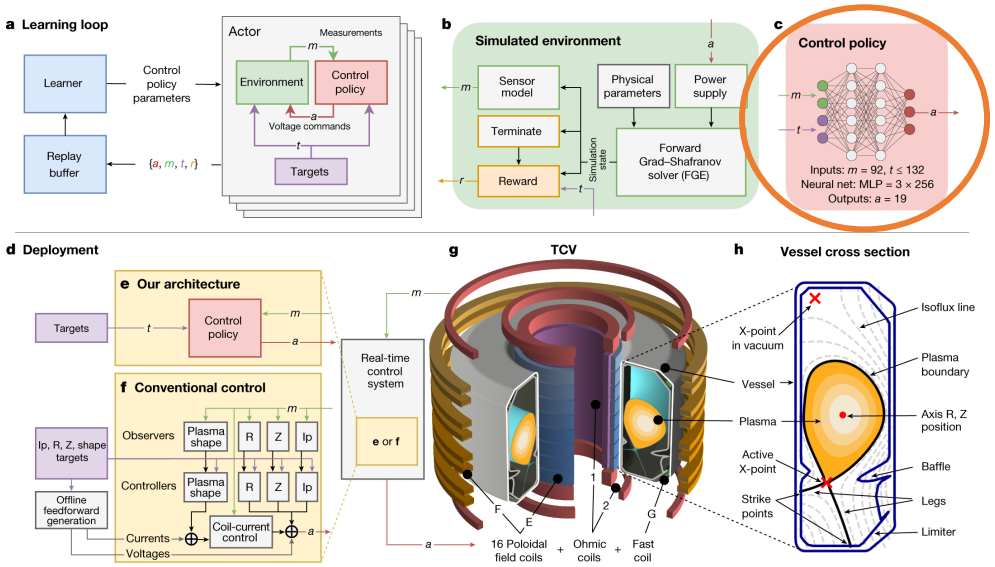
U.S. and European weather agencies are escalating their engagement with artificial intelligence as the technology rapidly advances

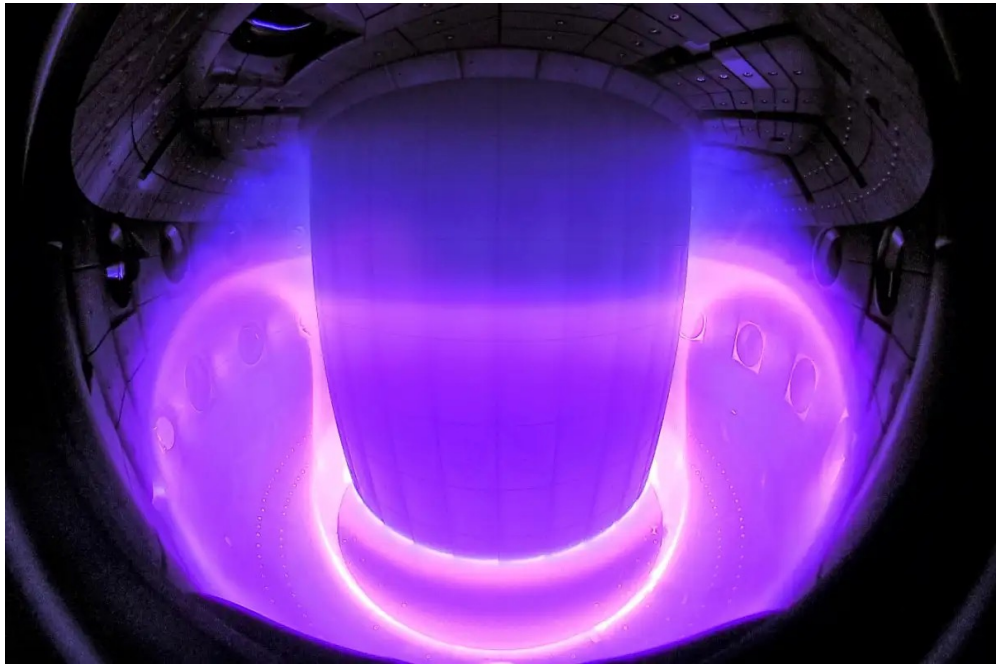


Pathak et al, FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, ArXiv (2022)

AI for science: a revolution?







Variable Configuration Tokamak (TCV) in Lausanne, Switzerland
Source: DeepMind & SPC/EPFL

nature

Explore content · About the journal · Publish with us

nature › articles › article

Article | Open access | Published: 16 February 2022

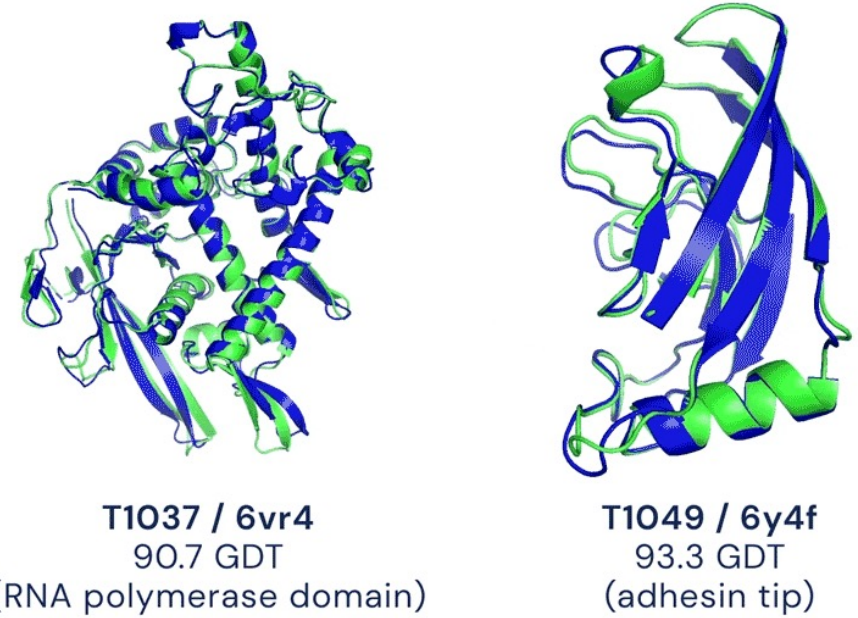
Magnetic control of tokamak plasmas through deep reinforcement learning

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, … Martin Riedmiller + Show authors

Nature **602**, 414–419 (2022) | Cite this article

206k Accesses | **223** Citations | **2430** Altmetric | Metrics

AI for science: a revolution?



nature

Explore content | About the journal | Publish with us

nature > articles > article

Article | Open Access | Published: 15 July 2021

Highly accurate protein structure prediction with AlphaFold

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, ... Demis Hassabis + Show authors

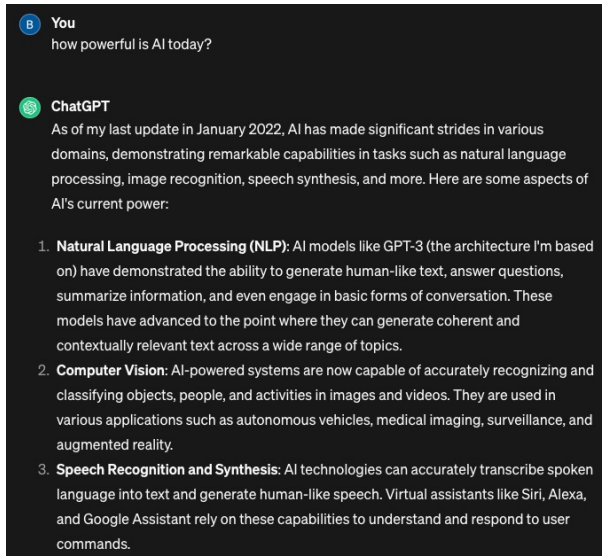
Nature **596**, 583–589 (2021) | Cite this article

958k Accesses | **5511** Citations | **3408** Altmetric | Metrics

Abstract

Proteins are essential to life, and understanding their structure can facilitate a mechanistic understanding of their function. Through an enormous experimental effort[1,2,3,4], the structures of around 100,000 unique proteins have been determined[5], but this represents a small fraction of the billions of known protein sequences[6,7]. Structural coverage is bottlenecked by the months to years of painstaking effort required to determine a single protein structure. Accurate computational approaches are needed to address this gap and to enable large-scale structural bioinformatics. Predicting the three-dimensional structure that a protein will adopt based solely on its amino acid sequence—the structure prediction component of the 'protein folding problem'[8]—has been an important open research problem for more than 50 years[9]. Despite recent progress[10,11,12,13,14], existing methods fall far short of atomic accuracy, especially when no homologous structure is available. Here we provide the first computational method that can regularly predict protein structures with atomic accuracy even in cases in which no similar structure is known. We validated an entirely redesigned version of our neural network-based model, AlphaFold, in the challenging 14th

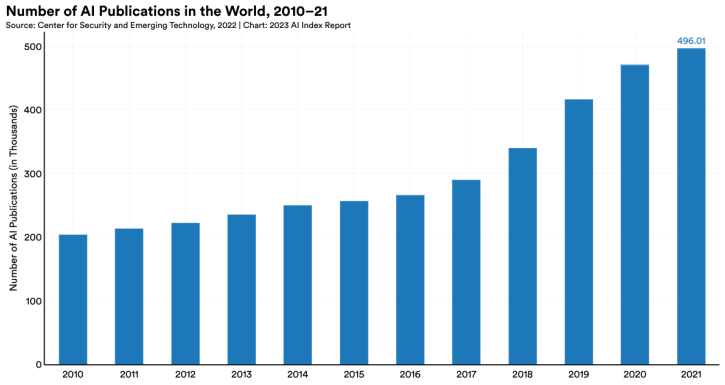
The rise of AI

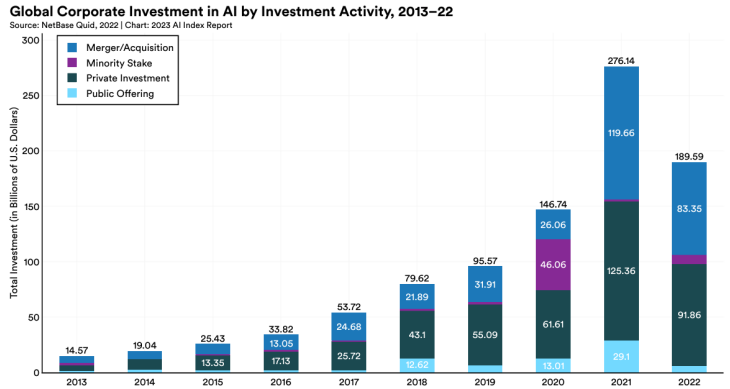


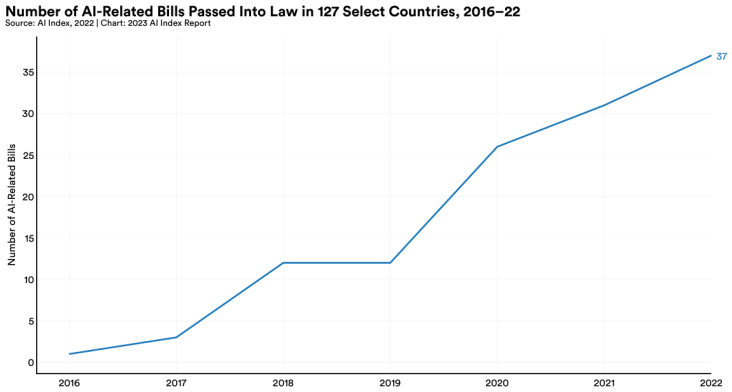
Brown et al, Language Models are Few-Shot Learners, NeurIPS (2020)

"a photograph of an astronaut riding a horse"

Source: Stable Diffusion Rombach et al, High-Resolution Image Synthesis with Latent Diffusion Models, CVPR (2022)

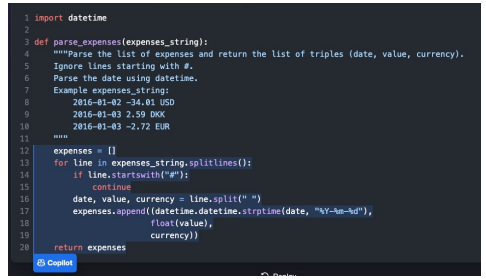




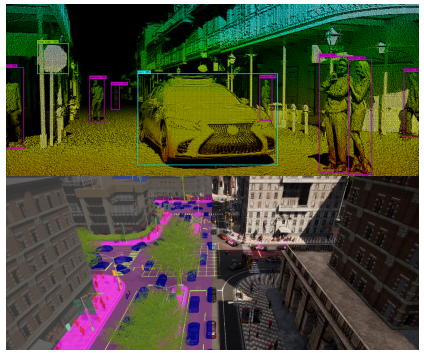


Source: AI Index Report, Stanford University

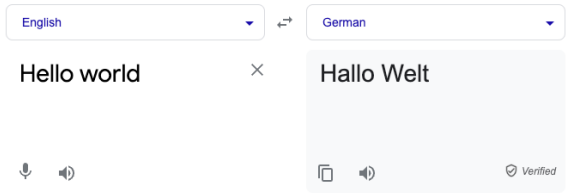
Reed et al., A Generalist Agent, TMLR (2022)



Source: GitHub Copilot

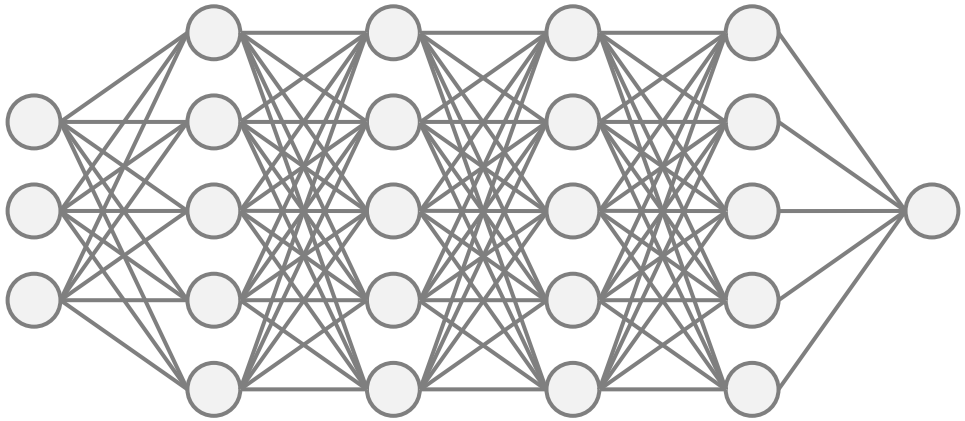


Source: Machine Learning for Autonomous Driving Workshop, NeurIPS (2023)



Source: Google Translate

What is deep learning?



Probability(Dog) = 1

Input

$\boldsymbol{x}$

Model

$NN(\boldsymbol{x}, \boldsymbol{\theta})$

Output

$\boldsymbol{y} = NN(\boldsymbol{x}, \boldsymbol{\theta})$

Deng et al, ImageNet: A large-scale hierarchical image database, CVPR (2009)

Neural networks are simply **flexible functions** fit to data

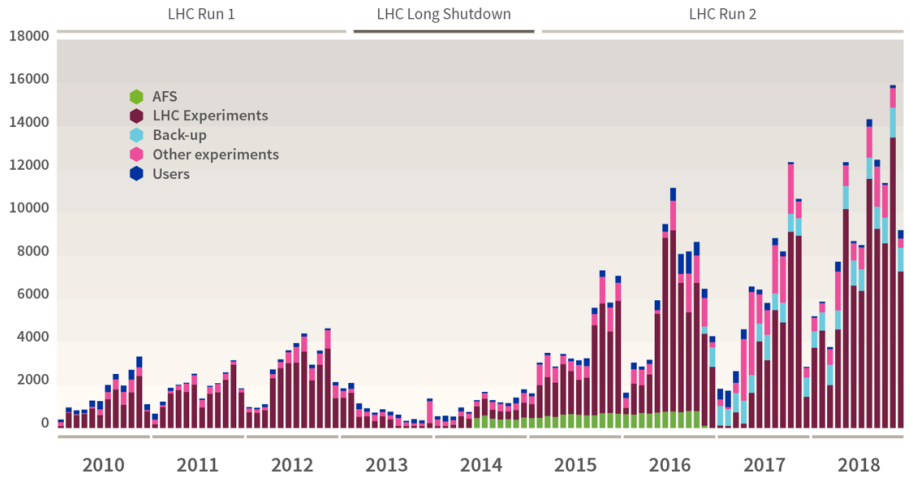
For example:

$$\boldsymbol{y} = W_2\sigma(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

Trained ($\boldsymbol{\theta}$ learned) using:
- Many examples of inputs and outputs
- A loss function
- An optimisation algorithm, e.g. stochastic gradient descent

Grand challenges in science

Data (in terabytes) recorded on tapes at CERN month-by-month (2010–2018) (Source: CERN)



~5,000 exoplanets discovered to date

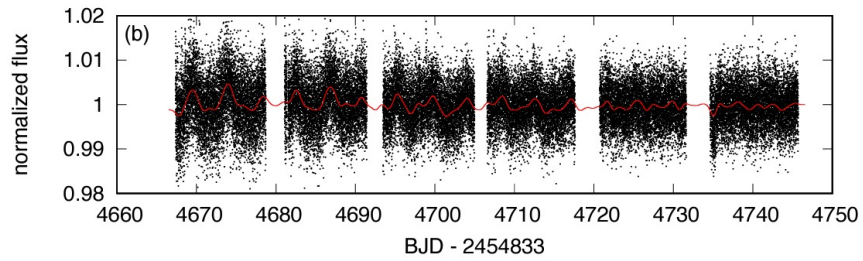
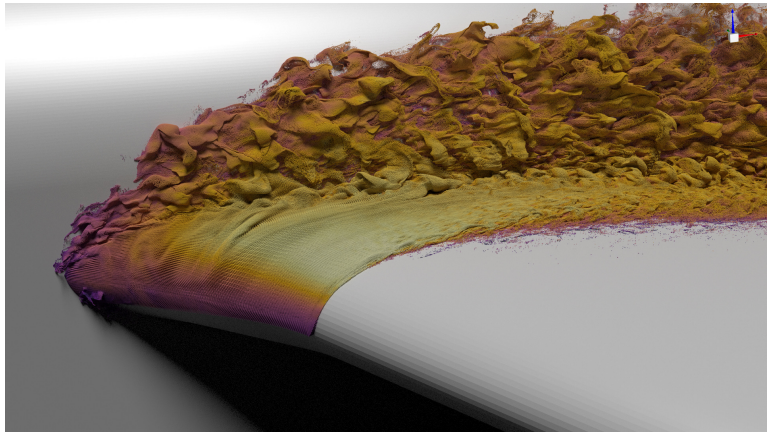


Figure 1. Light curves of K2-415 obtained by K2 (top; K2SFF) and TESS (bottom; PDC-SAP). Those data were taken at long (≈ 29 minutes) and short (2 minutes) cadences for K2 and TESS light curves, respectively. The red solid line in each panel represents the GP regression to the observed light curve (see Section 4.4).

Hirano et al, An Earth-sized Planet around an M5 Dwarf Star at 22 pc, The Astronomical Journal (2023)

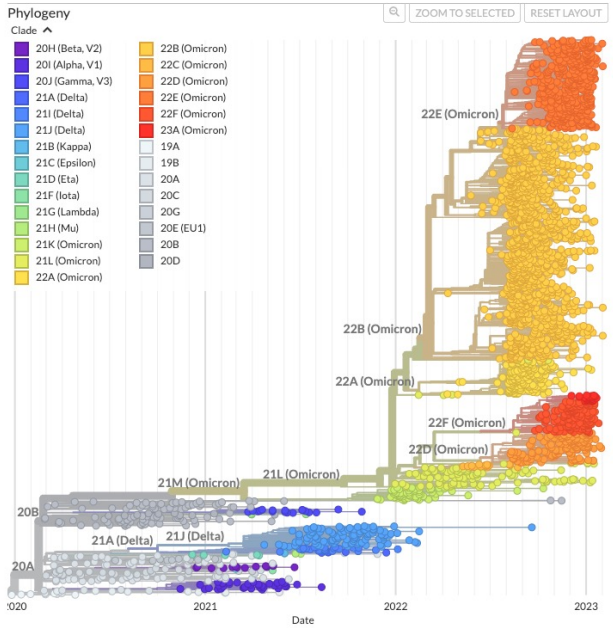
Air flow over a wing: wall-modeled large eddy simulation (~20 million core-hours) Source: NASA Ames

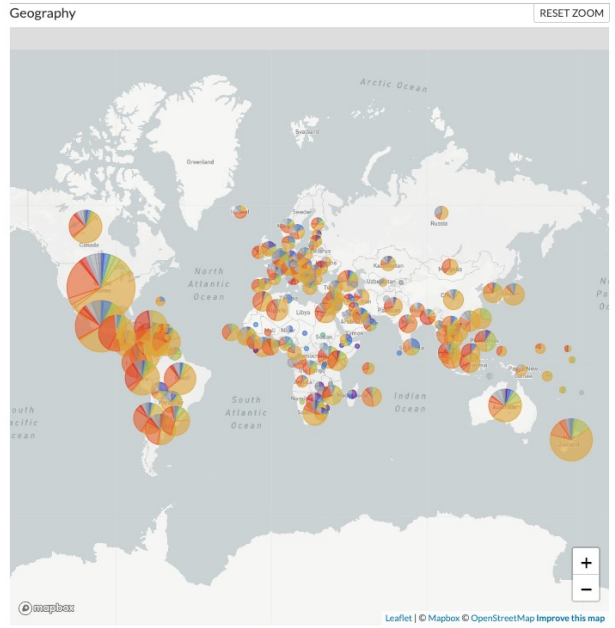


Genomic epidemiology of SARS-CoV-2 with subsampling focused globally over the past 6 months

Built with nextstrain/ncov. Maintained by the Nextstrain team. Enabled by data from GISAID.

Showing 2767 of 2767 genomes sampled between Dec 2019 and Feb 2023.

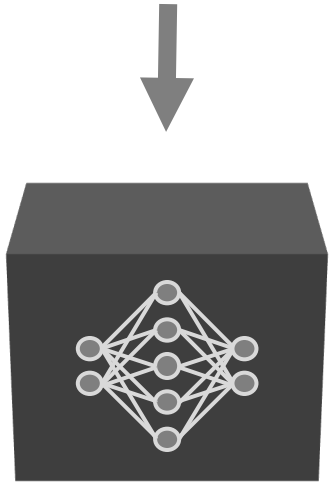




Source: Nextstrain

What is scientific machine learning?

Inputs to scientific workflow

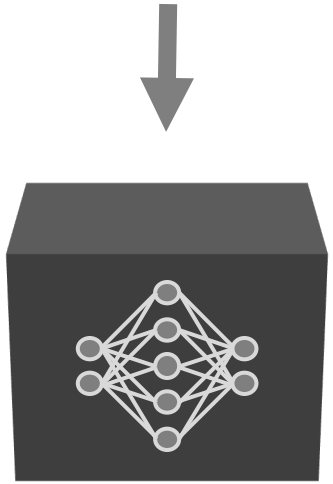


Black box model

Predictions

What is scientific machine learning?

Inputs to scientific workflow



Black box model

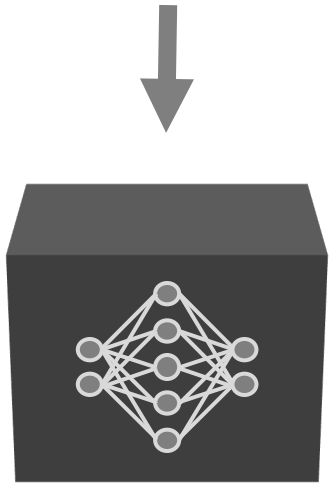
Predictions

Suffers from:

- Lack of **interpretability**
- Requires lots of training **data**
- Poor **generalisation**

What is scientific machine learning?

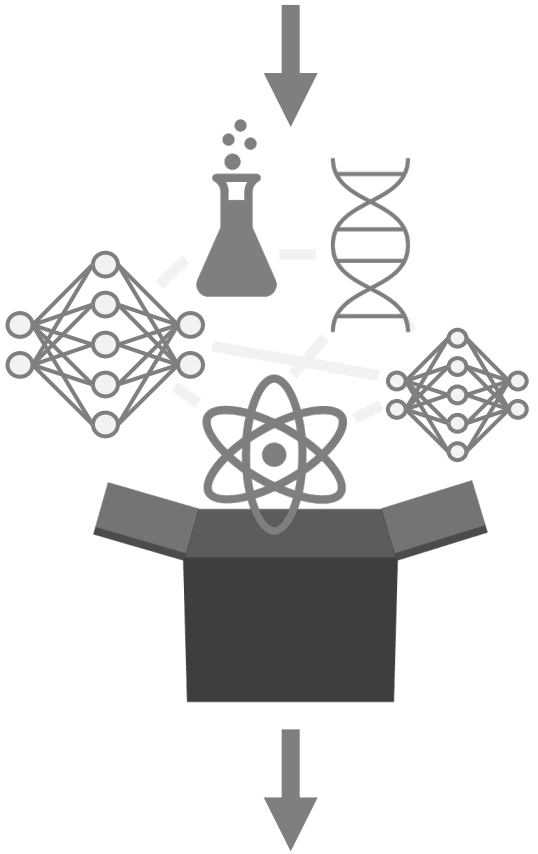
Inputs to scientific workflow



Black box model

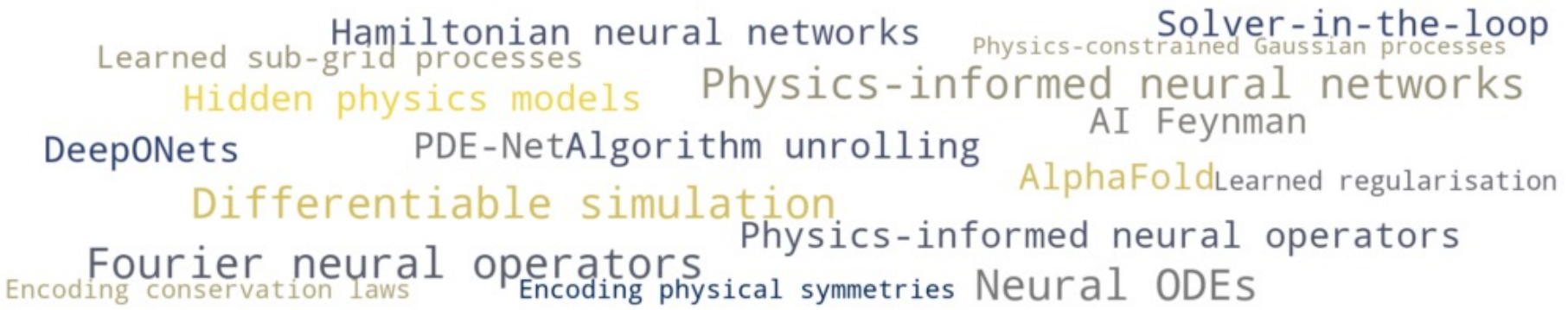
Predictions

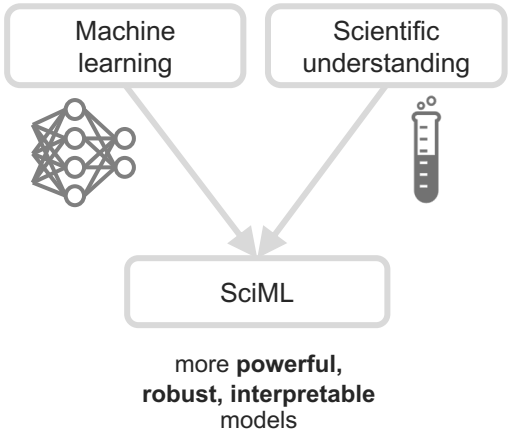
Inputs to scientific workflow



Predictions

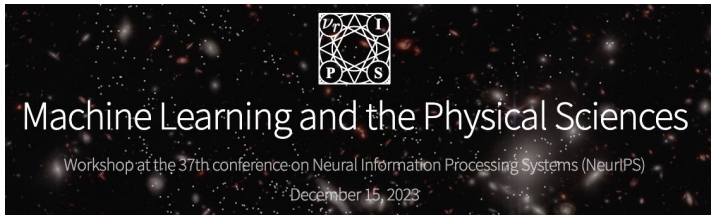
Scientific machine learning

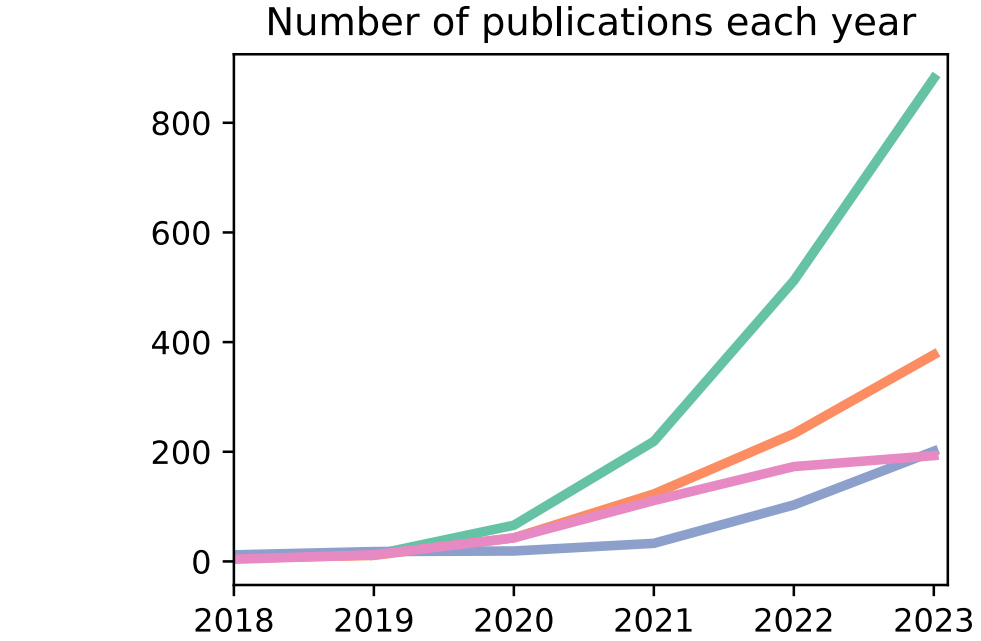




more **powerful, robust, interpretable** models

A rapidly growing field





Source: Scopus keyword search (Feb 2024)

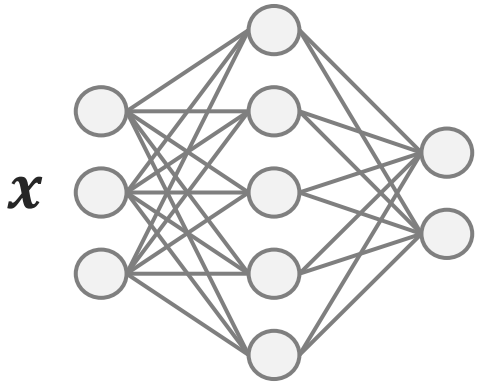
Introduction - summary

- AI has grown exponentially in the last decade
- It is driving many breakthroughs in **science,** and **changing** how we carry out scientific research
- SciML tightly **combines** scientific understanding with machine learning

Deep learning recap

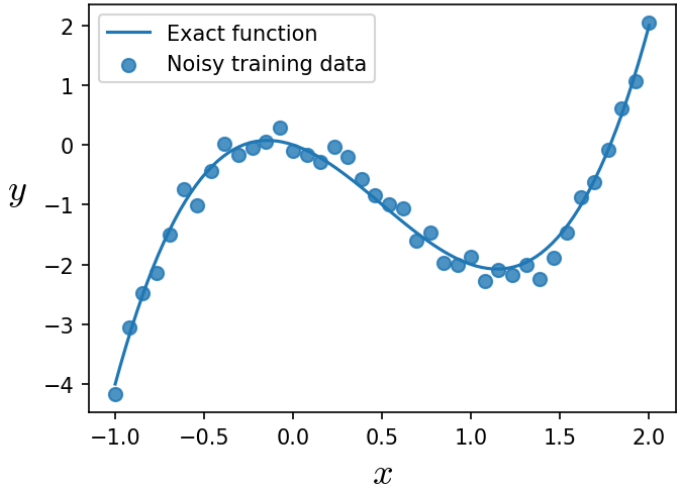
What is a neural network?

Neural networks are simply **flexible functions** fit to data



$$\boldsymbol{y} = NN(\boldsymbol{x}, \boldsymbol{\theta})$$

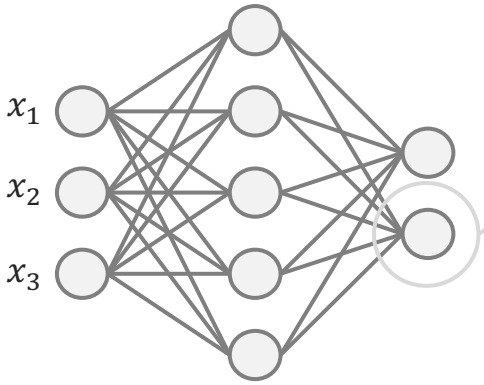
Example data:



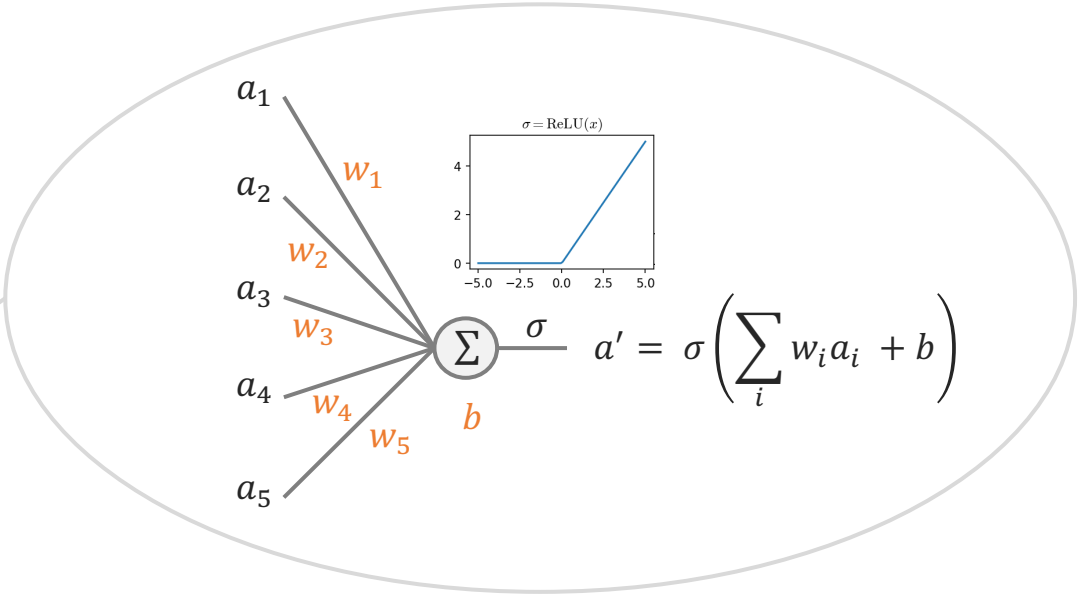
Goal: given training data, tune the parameters $\boldsymbol{\theta}$ so that the network approximates the true function, i.e.,

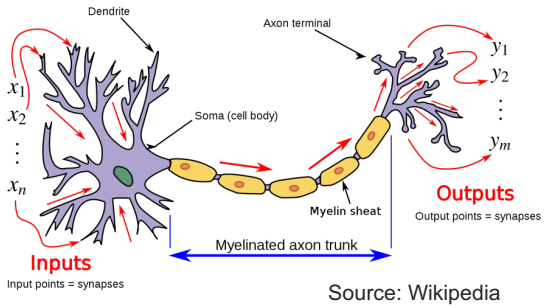
$$NN(\boldsymbol{x}, \boldsymbol{\theta}) \approx \boldsymbol{y}(\boldsymbol{x})$$

What is a neural network?



2-layer **multi-layer perceptron**

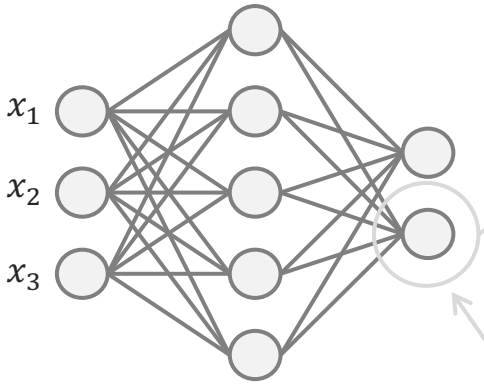




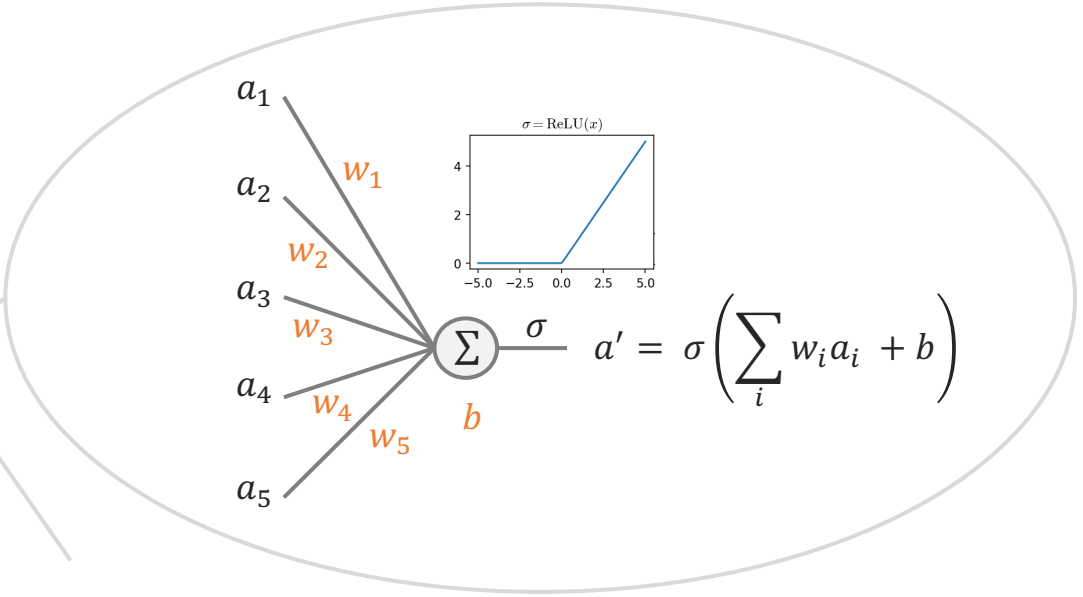
Source: Wikipedia

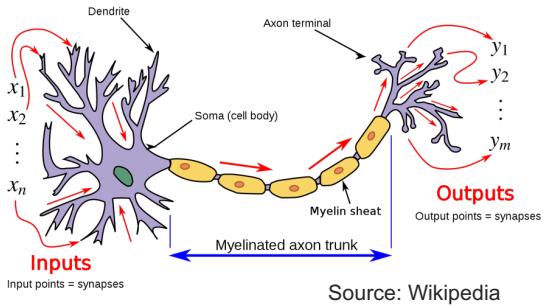
Biological neuron

What is a neural network?



2-layer **multi-layer perceptron**





Source: Wikipedia

Biological neuron

For last layer:

$$\begin{pmatrix} a_1' \\ a_2' \end{pmatrix} = \sigma\left(\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)$$

Entire network:

$$NN(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(W_2 \sigma(W_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)$$

How do we train neural networks?

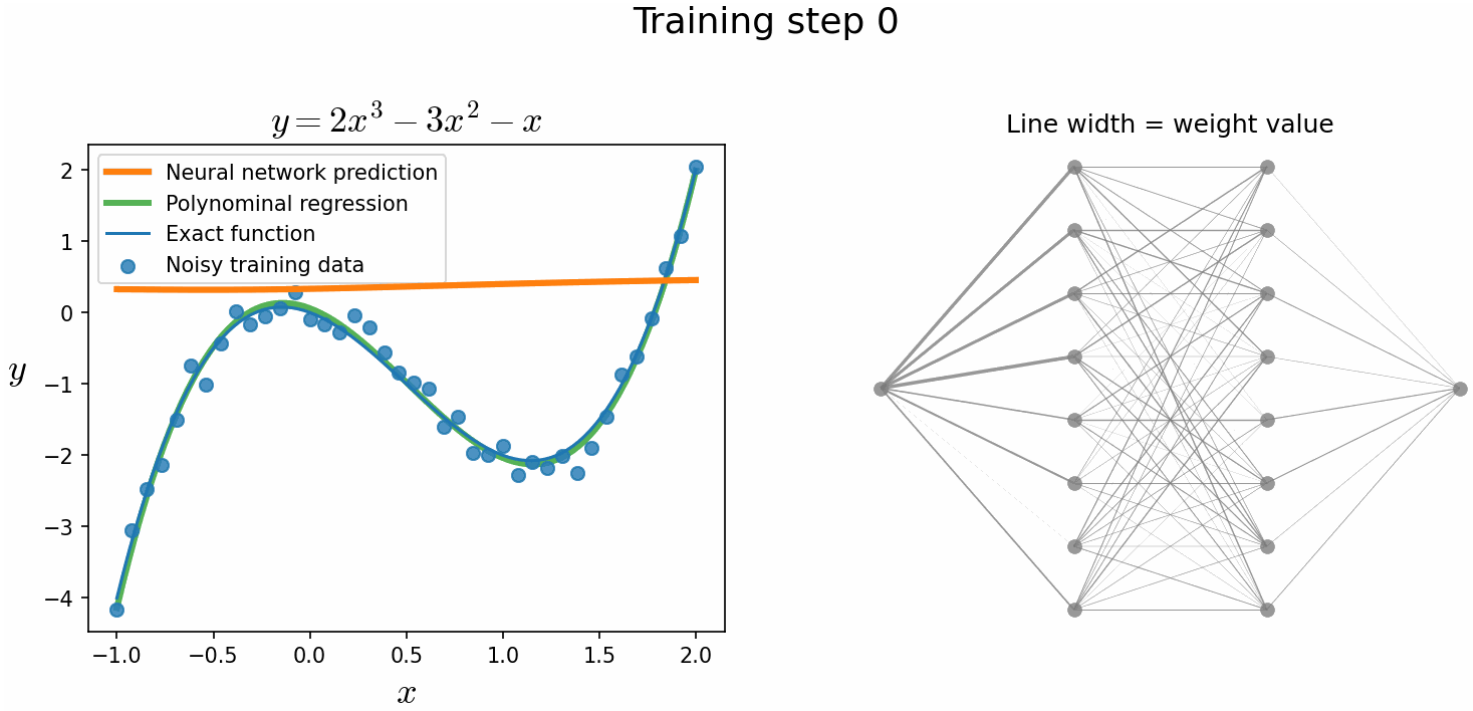
We tune the parameters so that they **minimise** some **loss function**, for example

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i}^{N}(NN(x_i, \boldsymbol{\theta}) - y_i)^2$$

Typically, by using **gradient descent**:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

γ = step size, e.g. 0.001

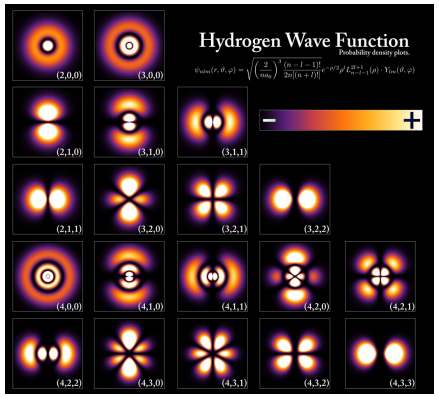


Lecture overview

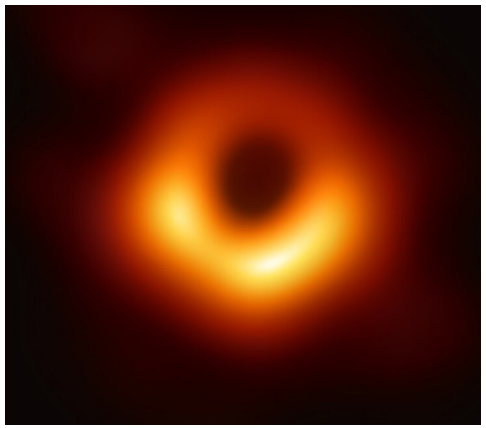
- Introduction to scientific machine learning (SciML)
- Rapid recap of deep learning fundamentals
- Introduction to physics-informed neural networks (PINNs)
- Live coding a PINN
- Current applications and limitations of PINNs

Introduction to PINNs

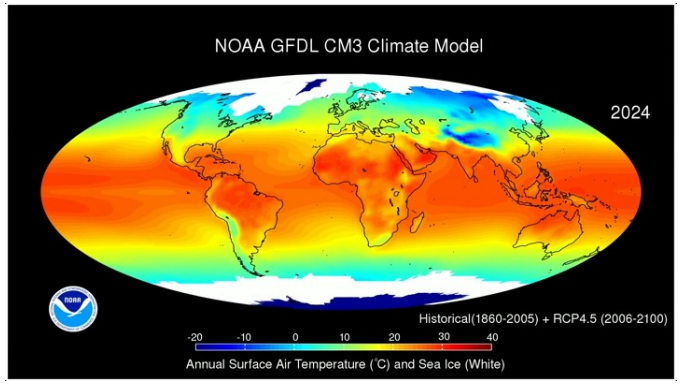
The importance of partial differential equations



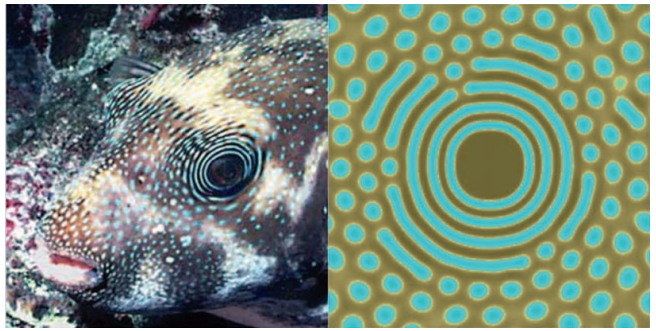
Source: Wikipedia



Source: The Event Horizon Telescope (2019)

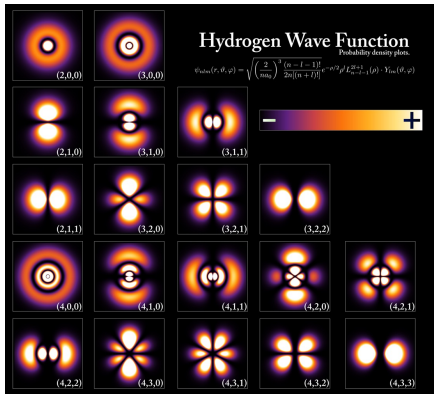


Source: NOAA



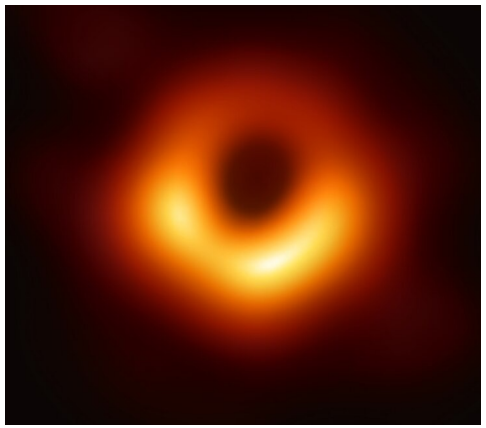
Source: Kondo and Miura, Science (2010)

The importance of partial differential equations



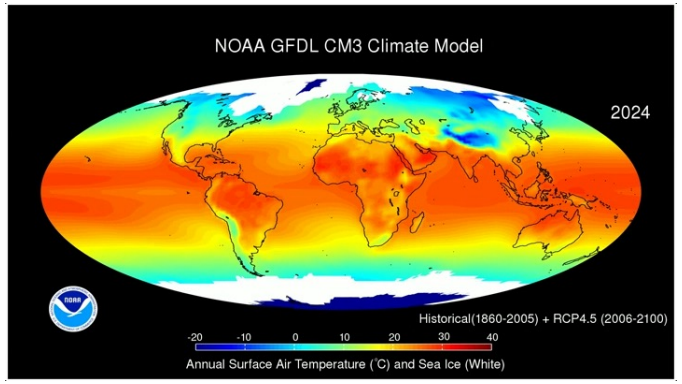
Source: Wikipedia

Schrödinger equation



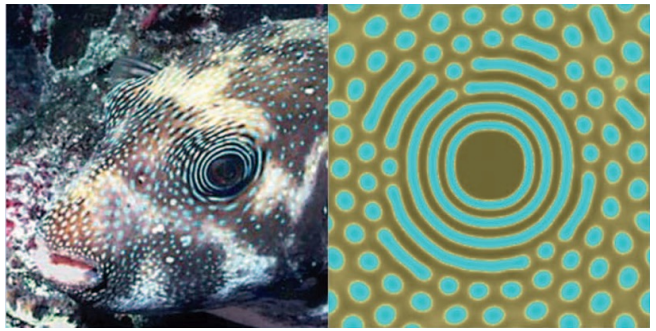
Source: The Event Horizon Telescope (2019)

Einstein field equations



Source: NOAA

Navier-Stokes equations



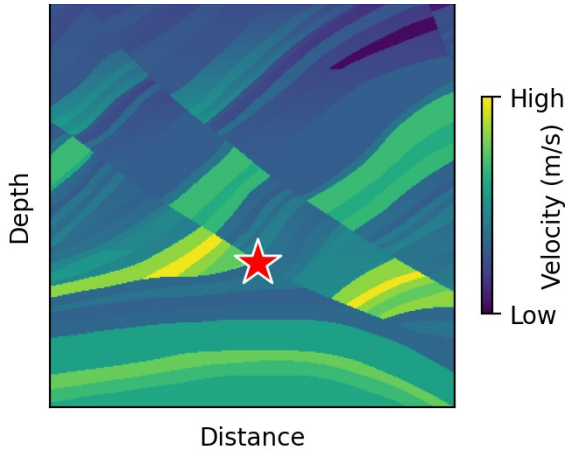
Source: Kondo and Miura, Science (2010)

Reaction-diffusion equation

Solving PDEs

Solving PDEs is:

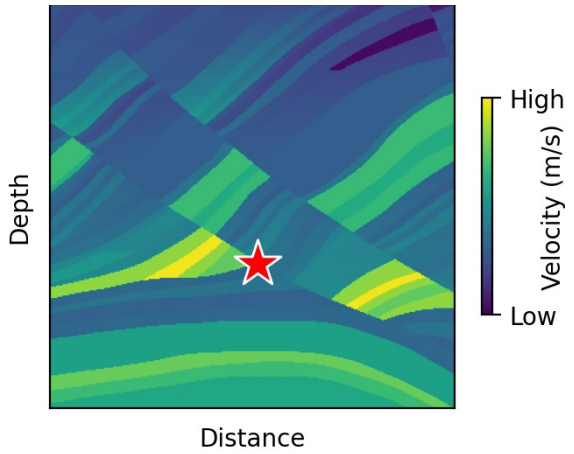
- Crucial for practically all domains of science
- Essential for understanding the behaviour of complex scientific phenomena



Solving PDEs

Solving PDEs is:

- Crucial for practically all domains of science
- Essential for understanding the behaviour of complex scientific phenomena



Wave equation:

$$\nabla^2 u - \frac{1}{c(x,y)^2}\frac{\partial^2 u}{\partial t^2} = s(x,y,t)$$

u = acoustic pressure
c = velocity
s = source function

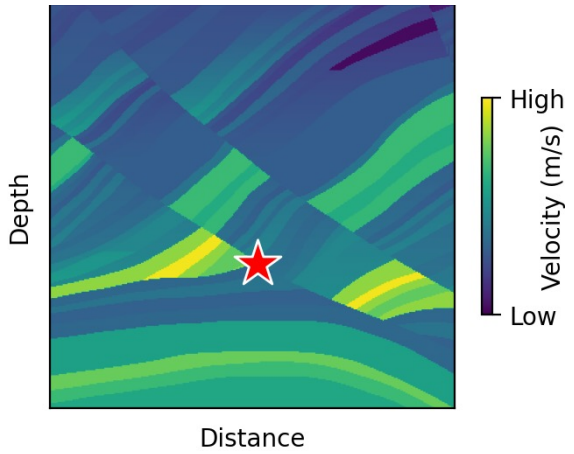
Initial conditions:

$$u(x,y,t=0) = 0$$
$$u_t(x,y,t=0) = 0$$

Solving PDEs

Solving PDEs is:

- Crucial for practically all domains of science
- Essential for understanding the behaviour of complex scientific phenomena



Wave equation:

$$\nabla^2 u - \frac{1}{c(x,y)^2}\frac{\partial^2 u}{\partial t^2} = s(x,y,t)$$

u = acoustic pressure
c = velocity
s = source function

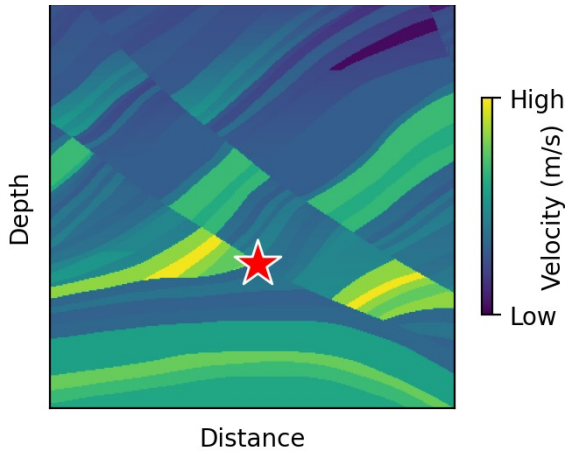
Initial conditions:

$$u(x,y,t=0) = 0$$
$$u_t(x,y,t=0) = 0$$

Physics-informed neural networks offer a way to **solve** PDEs

Using neural networks for simulation

Q: How could we solve this PDE using neural networks?



Wave equation:

$$\nabla^2 u - \frac{1}{c(x,y)^2}\frac{\partial^2 u}{\partial t^2} = s(x,y,t)$$

u = acoustic pressure
c = velocity
s = source function

Initial conditions:

$$u(x,y,t=0) = 0$$
$$u_t(x,y,t=0) = 0$$

What is a PINN?

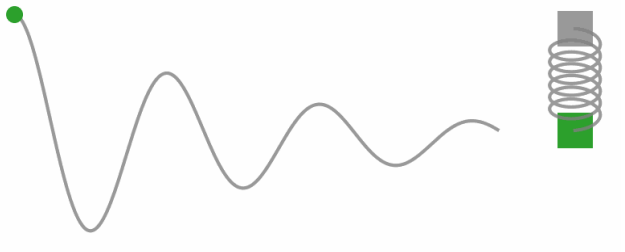
Damped harmonic oscillator:

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

Initial conditions:

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

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



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)

What is a PINN?

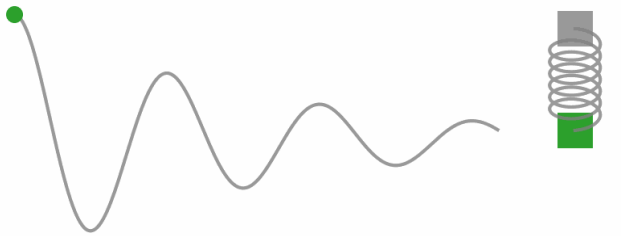
Damped harmonic oscillator:

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

Initial conditions:

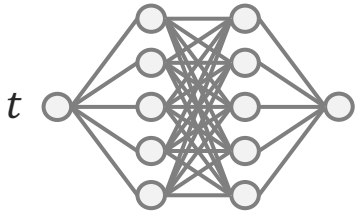
$$u(t=0) = 1$$
$$u_t(t=0) = 0$$

u = displacement
m = mass of oscillator
μ = coefficient of friction
k = spring constant



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

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



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

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)

What is a PINN?

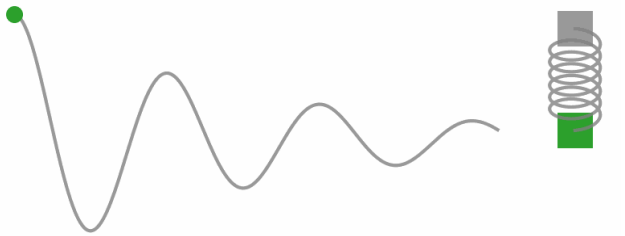
Damped harmonic oscillator:

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

Initial conditions:

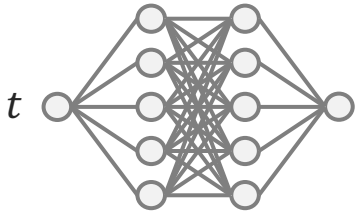
$$u(t=0) = 1$$
$$u_t(t=0) = 0$$

u = displacement
m = mass of oscillator
μ = coefficient of friction
k = spring constant



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

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



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

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)

What is a PINN?

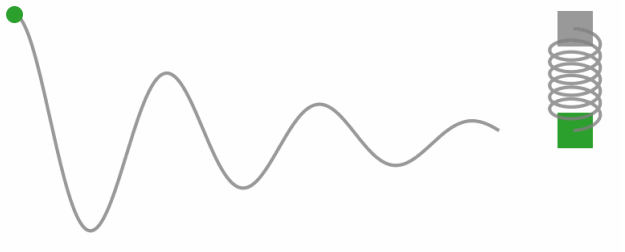
Damped harmonic oscillator:

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

Initial conditions:

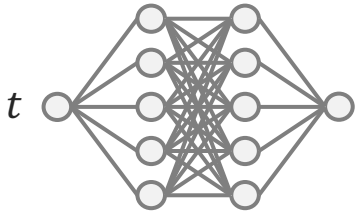
$$u(t=0) = 1$$
$$u_t(t=0) = 0$$

u = displacement
m = mass of oscillator
μ = coefficient of friction
k = spring constant



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

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



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

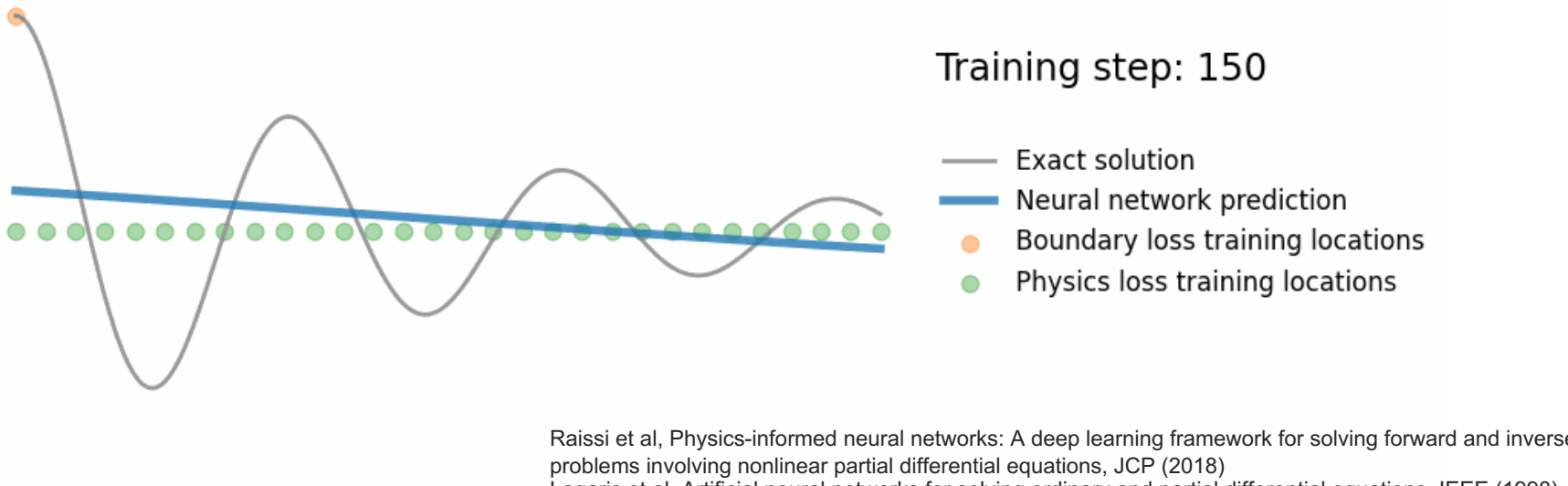
Train the network using the loss function:

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)

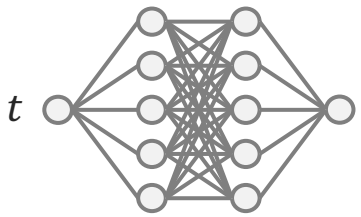
What is a PINN?

- $\{t_i\}_{i=1}^{N_p}$ are known as **collocation points**, which are sampled throughout the domain
- λ_1, λ_2 are scalar hyperparameters which **balance** the contribution of each loss term

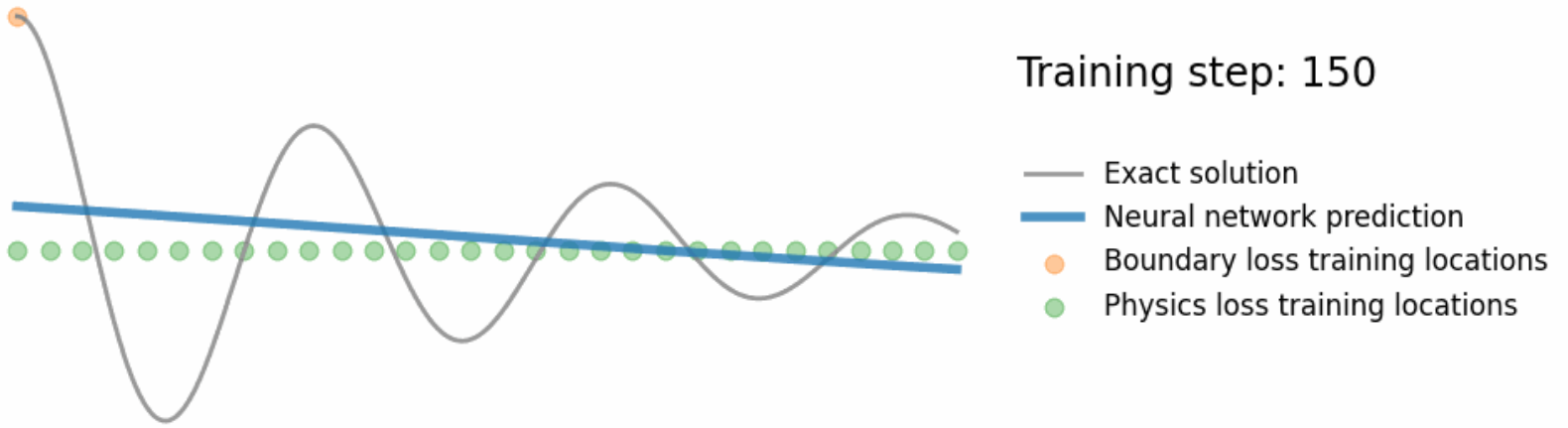
Train the network using the loss function:

$$\text{Boundary loss } L_b(\theta) \begin{cases} L(\theta) = \lambda_1 (NN(t=0;\theta) - 1)^2 \\ \quad + \lambda_2 \left(\frac{dNN}{dt}(t=0;\theta) - 0 \right)^2 \end{cases}$$

$$\text{Physics loss } L_p(\theta) \text{ (aka PDE residual)} \quad + \frac{1}{N_p} \sum_i^{N_p} \left(\left[m \frac{d^2}{dt^2} + \mu \frac{d}{dt} + k \right] NN(t_i;\theta) \right)^2$$



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



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)

PINNs are a general framework for solving PDEs

Given a PDE and its boundary/initial conditions

$$\mathcal{D}[u(x)] = f(x), \qquad x \in \Omega \subset \mathbb{R}^d$$
$$\mathcal{B}_k[u(x)] = g_k(x), \qquad x \in \Gamma_k \subset \partial\Omega$$

Where $\mathcal{D}$ is some differential operator, $\mathcal{B}_k$ are a set of boundary operators, and $u(x)$ is the solution to the PDE

PINNs train a neural network to **approximate** the solution to the PDE $NN(x;\theta) \approx u(x)$ using the following loss function:

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

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

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

PINNs are a general framework for solving PDEs

Given a PDE and its boundary/initial conditions

$$\mathcal{D}[u(x)] = f(x), \qquad x \in \Omega \subset \mathbb{R}^d$$
$$\mathcal{B}_k[u(x)] = g_k(x), \qquad x \in \Gamma_k \subset \partial\Omega$$

Where $\mathcal{D}$ is some differential operator, $\mathcal{B}_k$ are a set of boundary operators, and $u(x)$ is the solution to the PDE

PINNs train a neural network to **approximate** the solution to the PDE $NN(x;\theta) \approx u(x)$ using the following loss function:

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

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

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

For example, the 1+1D viscous Burgers' equation:

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

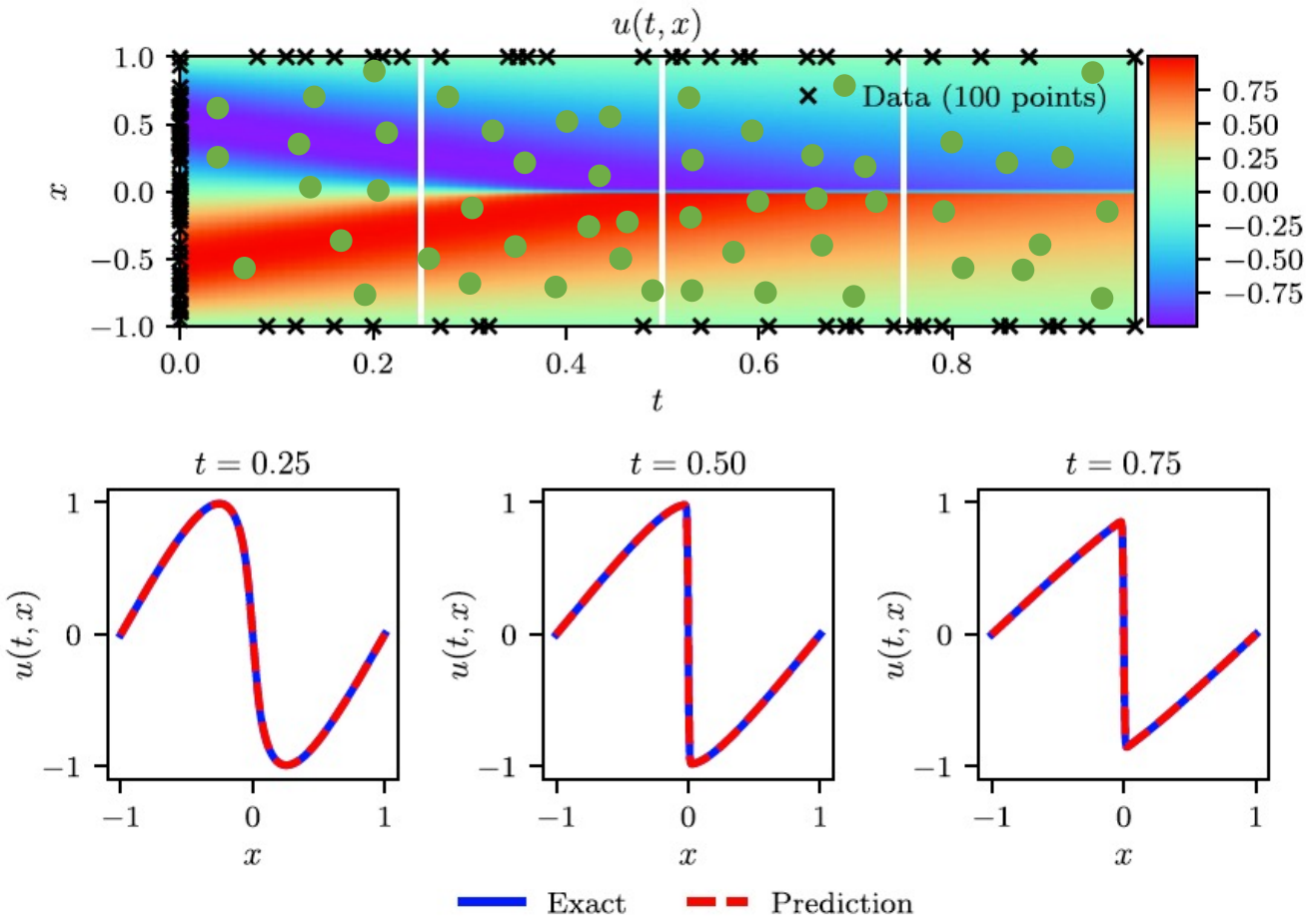
$$u(x,0) = -\sin(\pi x)$$
$$u(-1,t) = u(+1,t) = 0$$

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

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

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

PINNs for solving viscous Burgers' equation

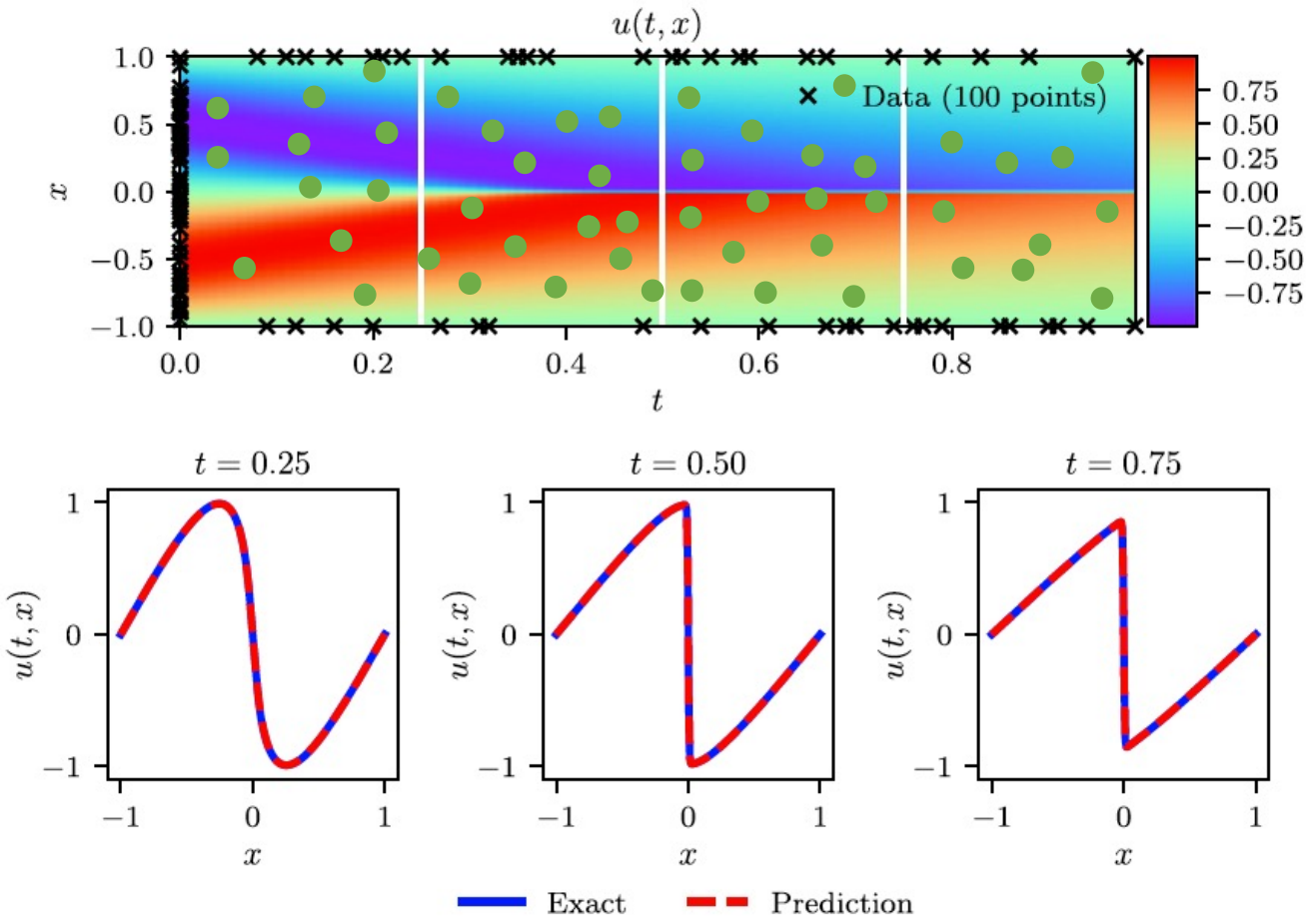


$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_{j}^{N_{b1}} \left(NN(x_j, 0; \theta) + \sin(\pi x_j) \right)^2 + \frac{\lambda_2}{N_{b2}} \sum_{k}^{N_{b2}} (NN(-1, t_k; \theta) - 0)^2 + \frac{\lambda_3}{N_{b3}} \sum_{l}^{N_{b3}} (NN(+1, t_l; \theta) - 0)^2$$

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

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

PINNs for solving viscous Burgers' equation



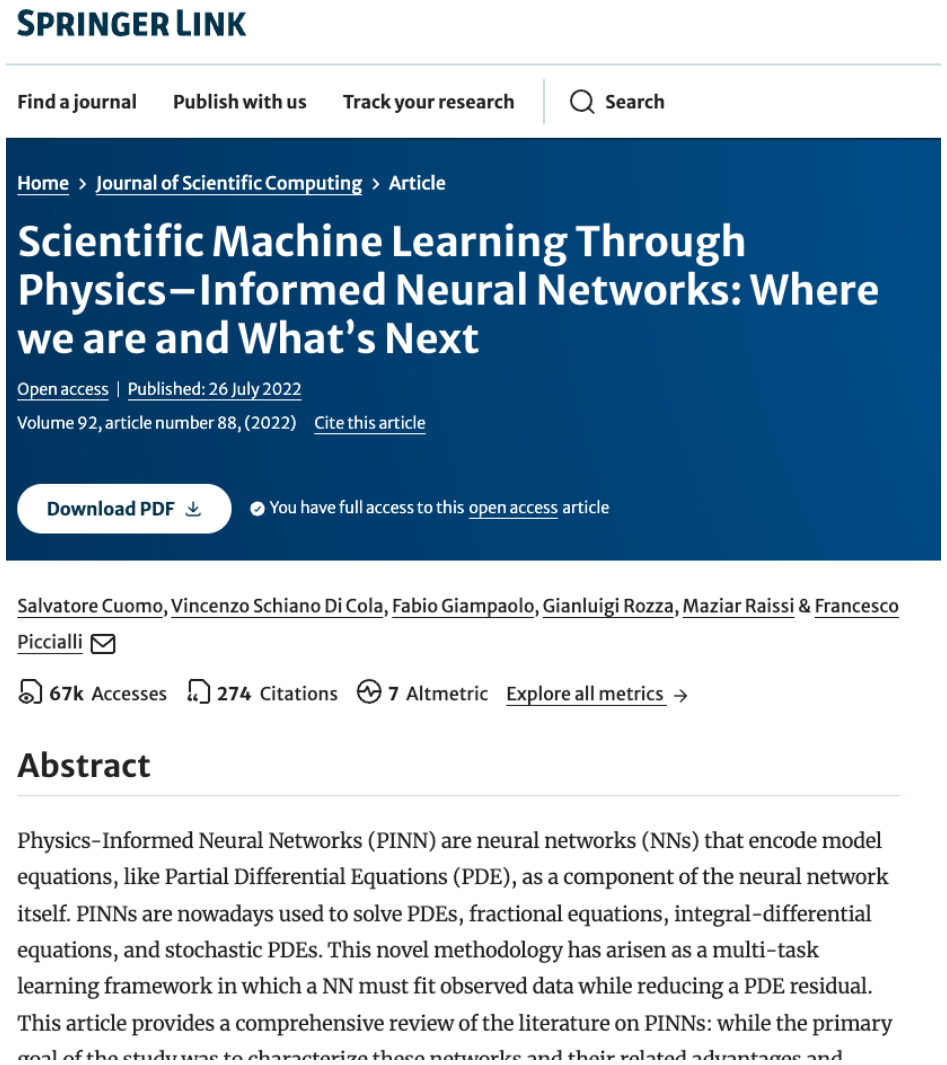
Raissi et al, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP (2018)

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

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

$\nu = 0.01/\pi$
$N_p = 10{,}000$ (Latin hypercube sampling)
$N_{b1} + N_{b2} + N_{b3} = 100$
Fully connected network with 9 layers, 20 hidden units (3021 free parameters)
Tanh activation function
L-BFGS optimiser

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

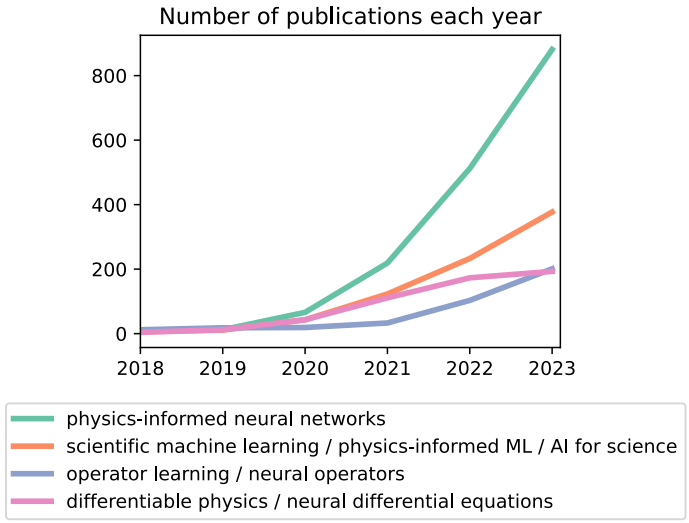
Download PDF | You have full access to this open access article

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

67k Accesses | 274 Citations | 7 Altmetric | Explore all metrics →

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

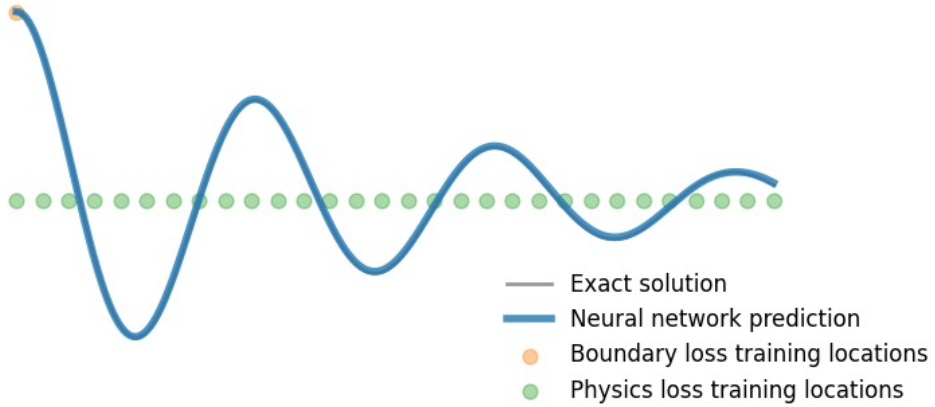


Source: Scopus keyword search (Feb 2024)

- The basic concepts behind PINNs were introduced in the 1990s (Lagaris et al, IEEE (1998) and others)
- Raissi et al, JCP (2018) is the seminal paper which reimplemented and extended PINNs using a modern deep learning framework

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)

PINN training loop

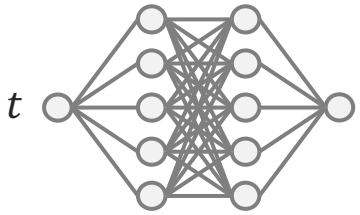


Boundary loss $L_b(\theta)$

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

Physics loss $L_p(\theta)$

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



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

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**
4. Compute loss
5. Compute gradient of loss function **with respect to network parameters**
6. Take gradient descent step

How can we compute the gradients (e.g. $\frac{dNN}{dt}$ and $\frac{dL}{d\theta}$) required in steps 3 and 5?

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]
```

Live-coding a PINN in PyTorch

Introduction to PINNs - summary

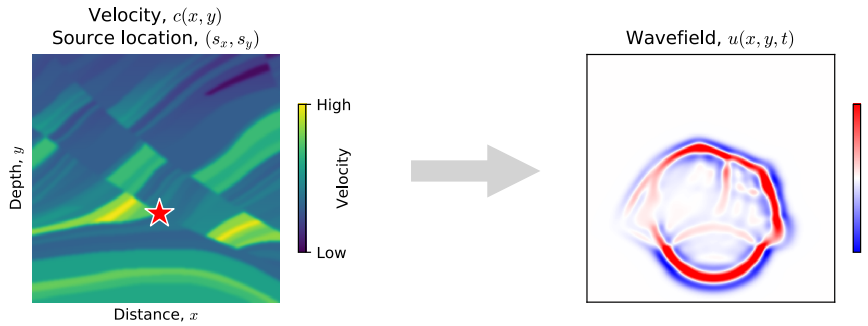
- PINNs are an alternative approach for solving **forward, inverse** and **equation discovery** problems related to differential equations
- They are **mesh-free**
- They use a neural network to **directly approximate** the solution to the PDE

Lecture overview

- Introduction to scientific machine learning (SciML)
- Rapid recap of deep learning fundamentals
- Introduction to physics-informed neural networks (PINNs)
- Live coding a PINN
- Current applications and limitations of PINNs

Applications of PINNs

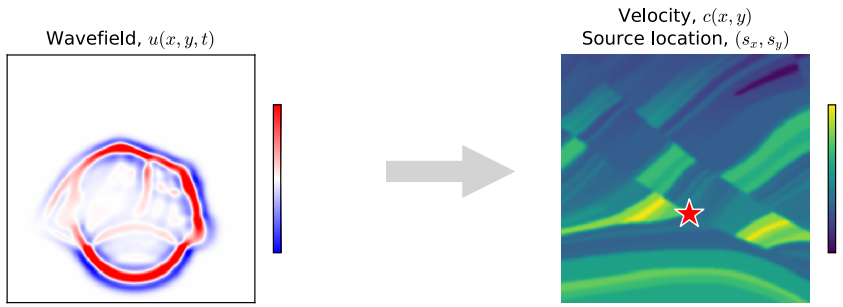
Key scientific tasks



Forward simulation

Estimate wavefield $u(x, y, t)$
Given velocity $c(x, y)$
and source location (s_x, s_y)

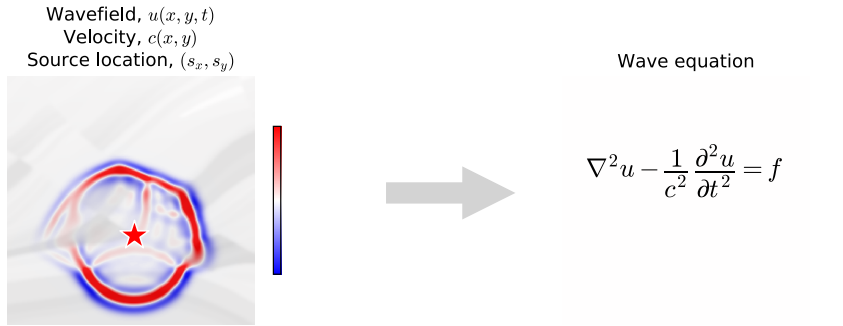
$$b = F(a)$$



Inversion

Estimate velocity $c(x, y)$
and source location (s_x, s_y)
Given wavefield $u(x, y, t)$

$$b = F(a)$$



Equation discovery

Estimate governing equation
Given wavefield $u(x, y, t)$,
velocity $c(x, y)$,
and source location (s_x, s_y)

$$b = F(a)$$

PINNs can be used to solve **forward**, **inverse** and **equation discovery** problems related to PDEs

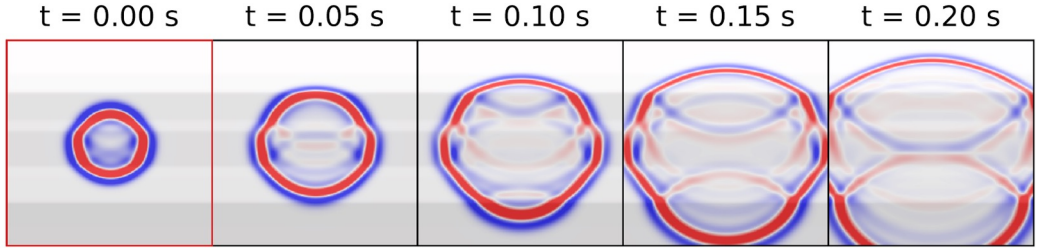
a = set of input conditions

F = physical model of the system (usually a PDE)

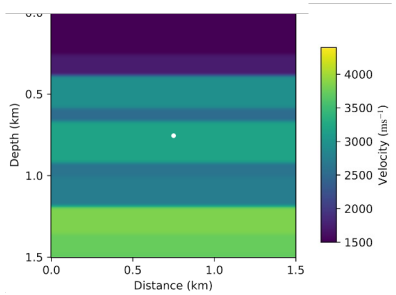
b = resulting properties given F and a

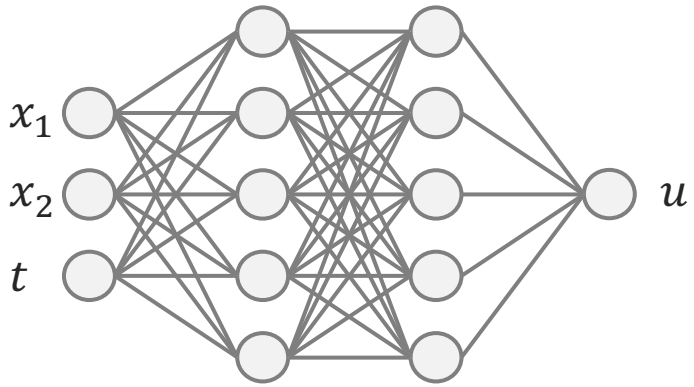
PINNs for solving wave equation

Ground truth FD simulation



Velocity model, $c(x)$





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

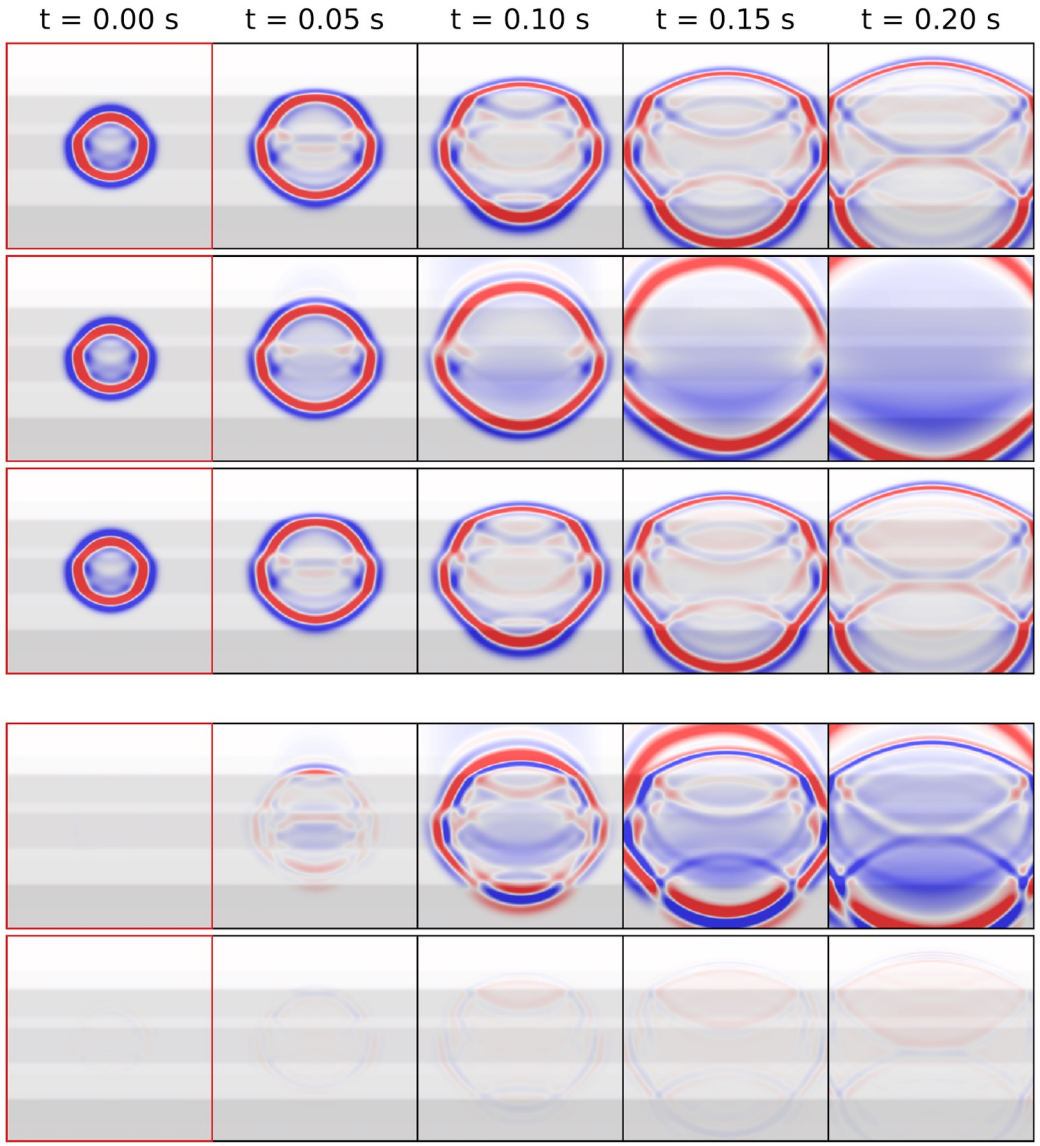
$$L_b(\theta) = \frac{\lambda}{N_b}\sum_{j}^{N_b}\left(NN(x_j, t_j; \theta) - u_{FD}(x_j, t_j)\right)^2$$

Boundary data from FD simulation (first 0.02 seconds)

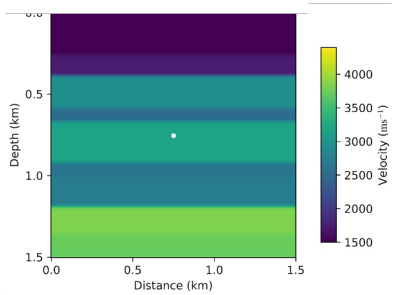
$$L_p(\theta) = \frac{1}{N_p}\sum_{i}^{N_p}\left(\left[\nabla^2 - \frac{1}{c(x_i)^2}\frac{\partial^2}{\partial t^2}\right] NN(x_i, t_i; \theta)\right)^2$$

Collocation points randomly sampled over entire domain (up to 0.2 seconds)

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

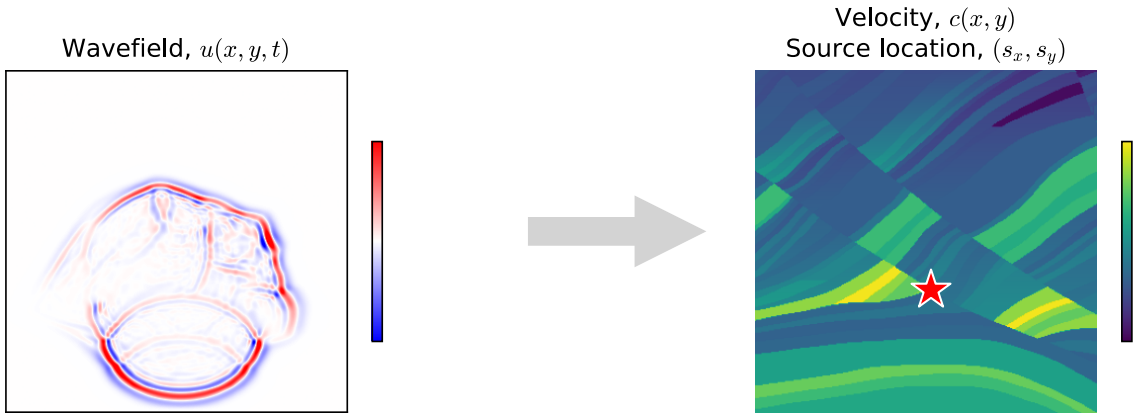
Training time: **~1 hour**

PINNs for inverse problems

What is an inverse problem?

Wave equation:

$$\nabla^2 u - \frac{1}{c(x)^2}\frac{\partial^2 u}{\partial t^2} = 0$$



$b =$ observed wavefield $u(x, t)$

$a =$ (velocity model $c(x)$, source location)

- Fundamentally, inverse problems are **search** problem
- It is often useful to frame them as an optimisation problem, for example:

$$\min_{\hat{a}} \|b - F(\hat{a})\|^2$$

$$b = F(a)$$

$a =$ set of input conditions

$F =$ physical model of the system

$b =$ resulting properties given F and a

PINNs for inversion

PINNs for solving **forward** simulation:

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

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

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

For example:

$$D = \left[\nabla^2 - \frac{1}{c(x)^2}\frac{\partial^2}{\partial t^2}\right]$$
$$f = 0$$

PINNs for inversion

PINNs for solving **forward** simulation:

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

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

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

For example:

$$D = \left[\nabla^2 - \frac{1}{c(x)^2}\frac{\partial^2}{\partial t^2}\right]$$
$$f = 0$$

PINNs for solving **inverse** problems:

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

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

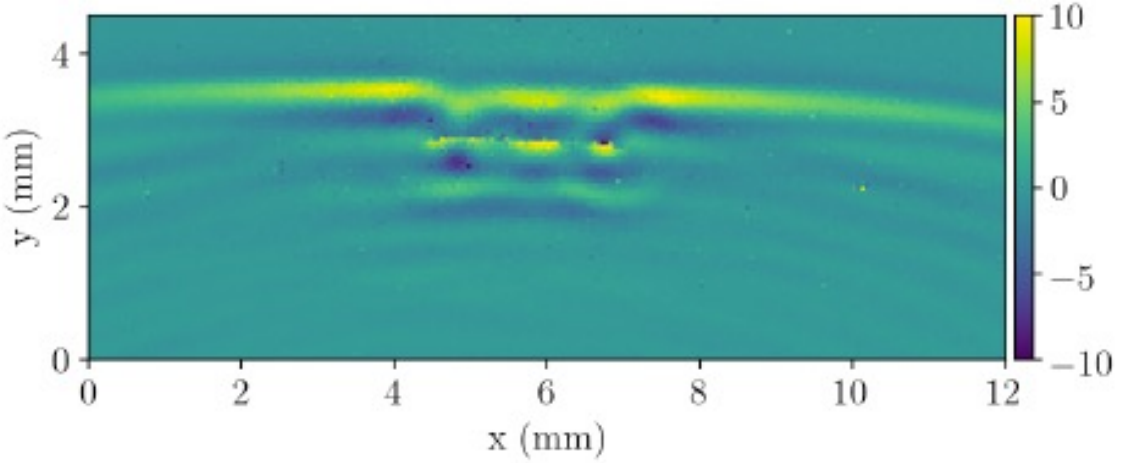
$$L_d(\theta) = \frac{\lambda}{N_d} \sum_l^{N_d} \|NN(x_l;\theta) - u_l\|^2 \quad \textbf{Data loss}$$

Where ϕ are unknown, underlying PDE parameters we wish to invert for, and $\{x_l, u_l\}$ are a set of (potentially noisy) observational data

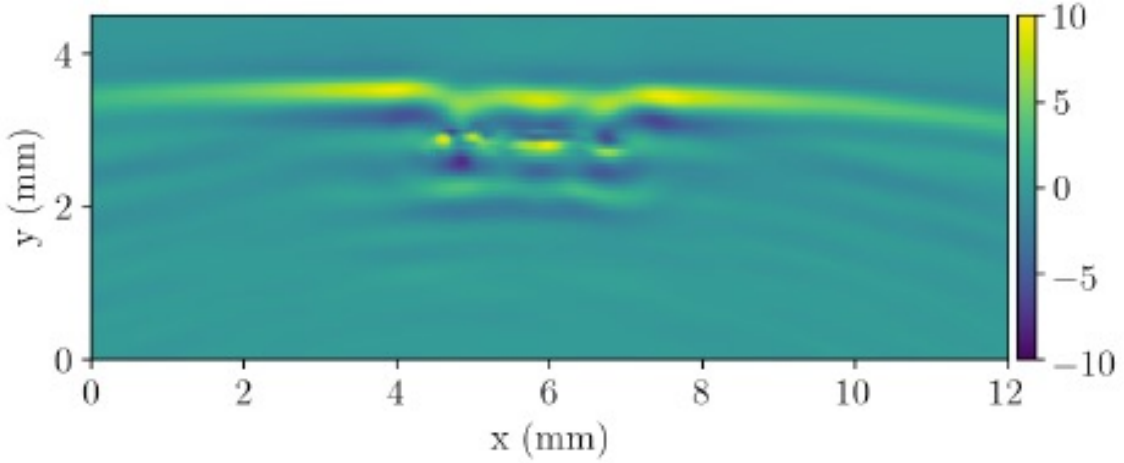
We **simultaneously** learn θ and ϕ when training the PINN

PINNs for wave equation

Shukla et al, Physics-Informed Neural Network for Ultrasound Nondestructive Quantification of Surface Breaking Cracks, Journal of Nondestructive Evaluation (2020)



(a) Actual data at $t = 12.38\ \mu s$.

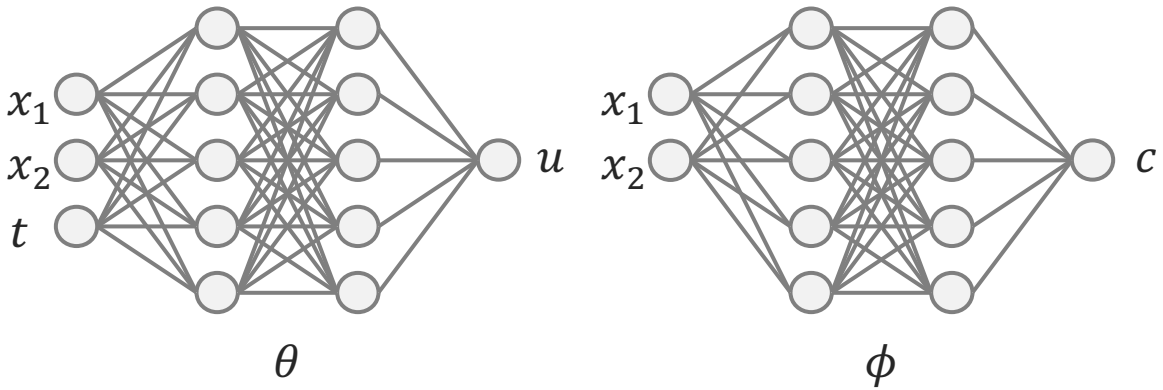


(b) Data recovered from PINN simulation at $t = 12.38\ \mu s$.

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

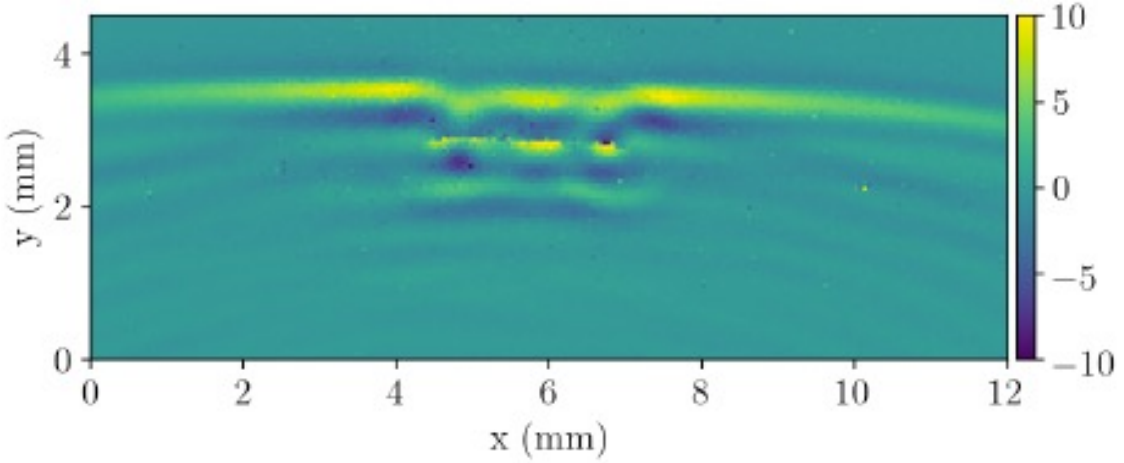
$$L_d(\theta) = \frac{\lambda}{N_d} \sum_{l}^{N_d} (NN(x_l, t_l; \theta) - u_{obs\,l})^2$$

Treat velocity model as **another** neural network, and simultaneously learn it

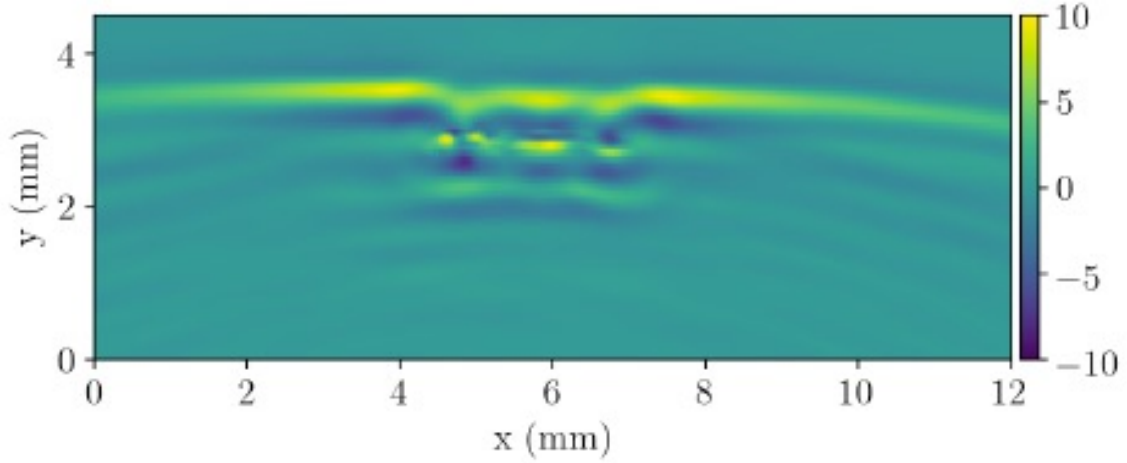


PINNs for wave equation

Shukla et al, Physics-Informed Neural Network for Ultrasound Nondestructive Quantification of Surface Breaking Cracks, Journal of Nondestructive Evaluation (2020)



(a) Actual data at $t = 12.38\ \mu s$.

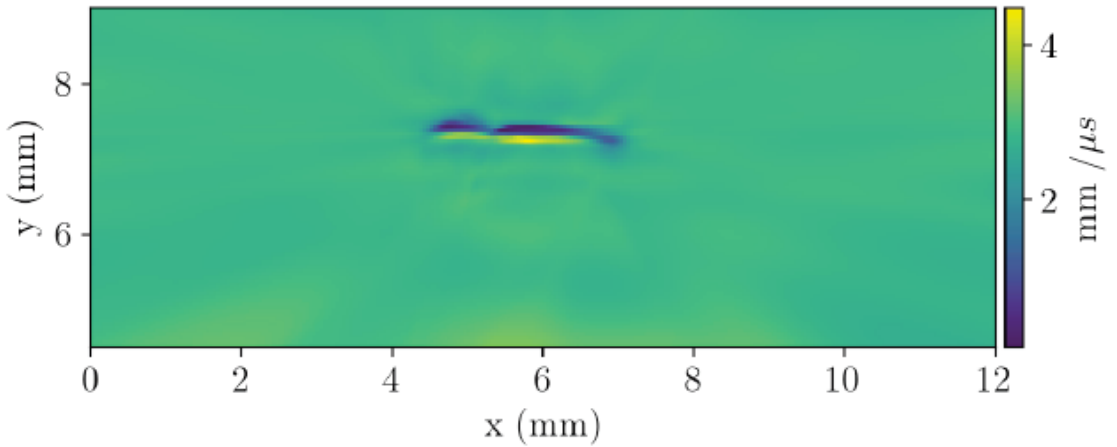


(b) Data recovered from PINN simulation at $t = 12.38\ \mu s$.

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

$$L_d(\theta) = \frac{\lambda}{N_d} \sum_{l}^{N_d} (NN(x_l, t_l; \theta) - u_{obs\,l})^2$$

Treat velocity model as **another** neural network, and simultaneously learn it



(d) Speed $v(x, y)$ recovered from PINN simulation.

Live-coding a PINN in PyTorch

Limitations of PINNs

Advantages / limitations of PINNs

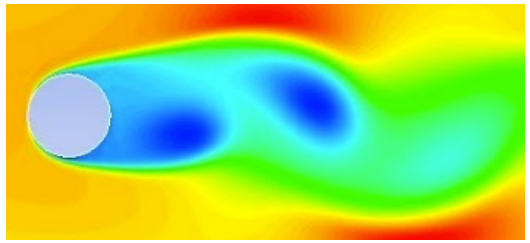
Advantages

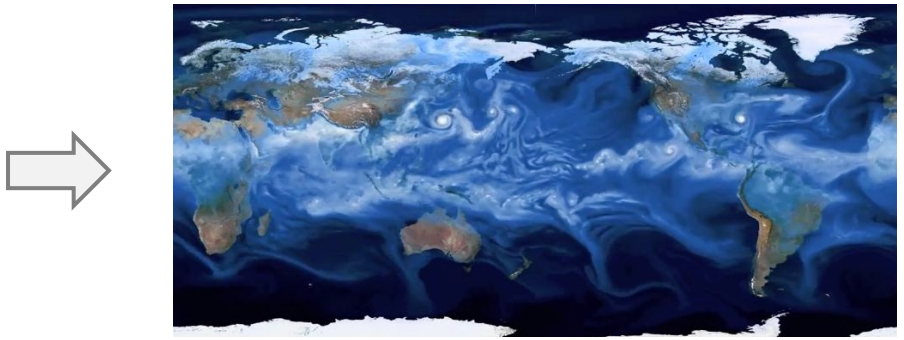
- **Mesh-free**
- Can jointly solve **forward** and **inverse** problems
- Often performs well on “messy” problems (where some observational data is available)
- Tractable, analytical solution gradients (e.g. for sensitivity analysis)
- Mostly **unsupervised**

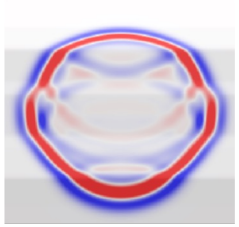
Limitations

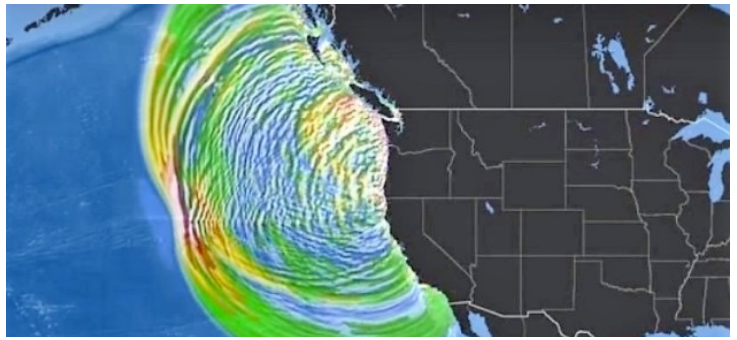
- **?**

Scaling to more complex problems





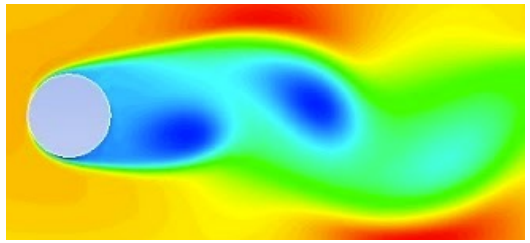


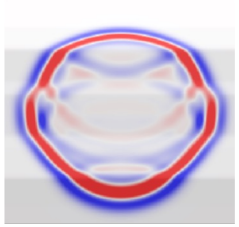


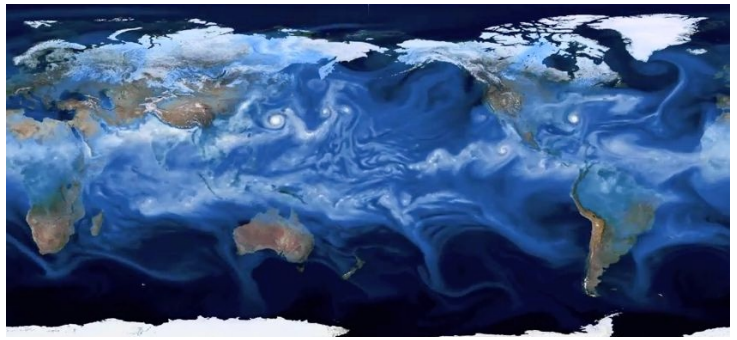
Majority of PINN research focuses on **toy/simplified** problems,
as proof-of-principle studies

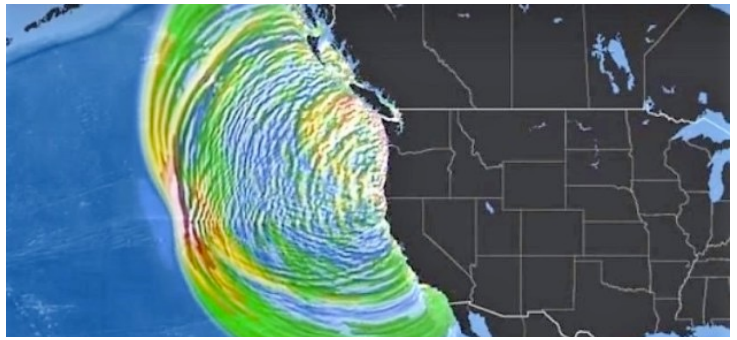
Image credits: Lawrence Berkeley National Laboratory / NOAA / NWS / Pacific Tsunami Warning Center

Scaling to more complex problems









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

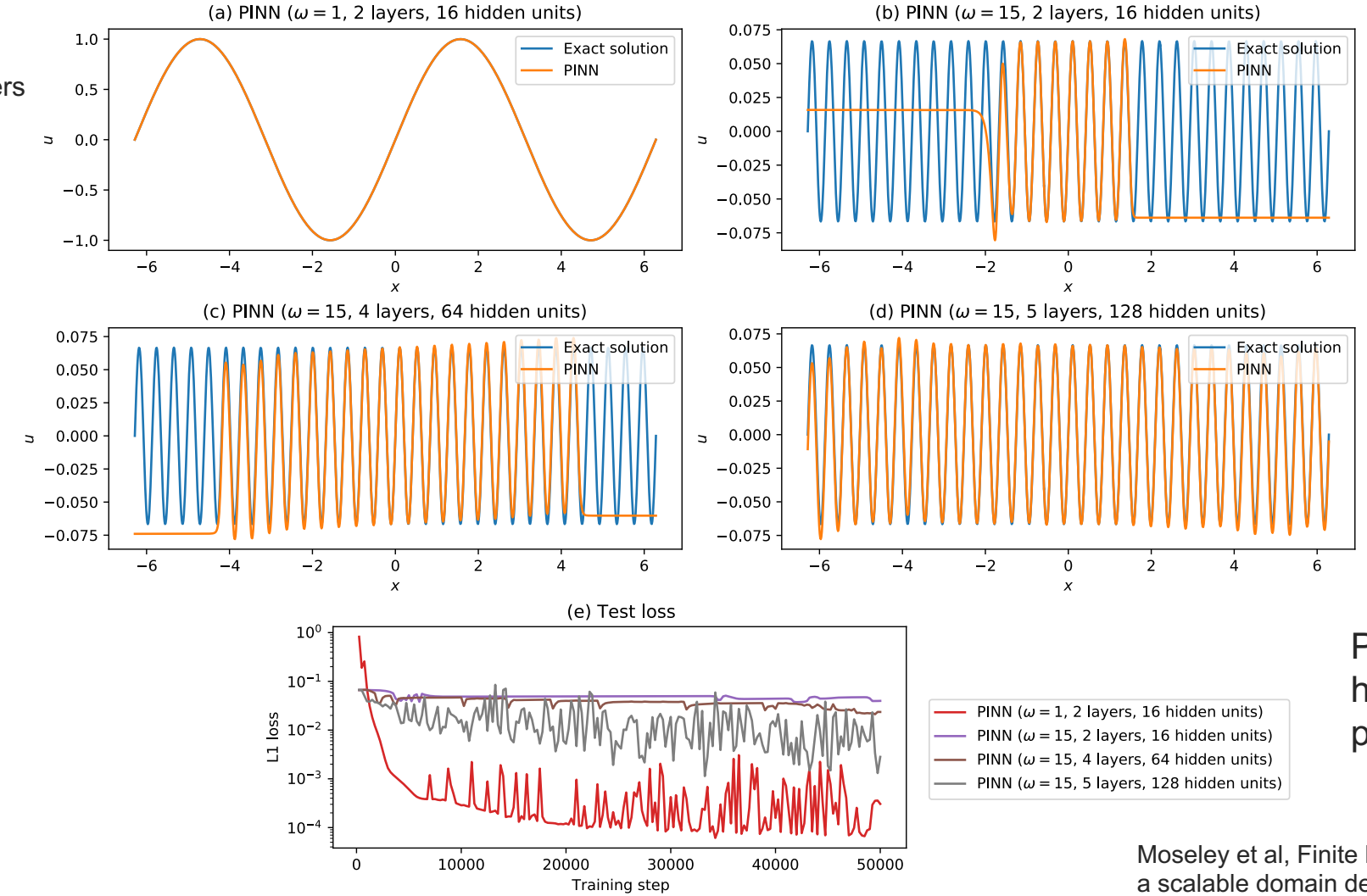
How do PINNs cope in this setting?

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 PINNs to higher frequencies

321 free parameters



PINN solving:

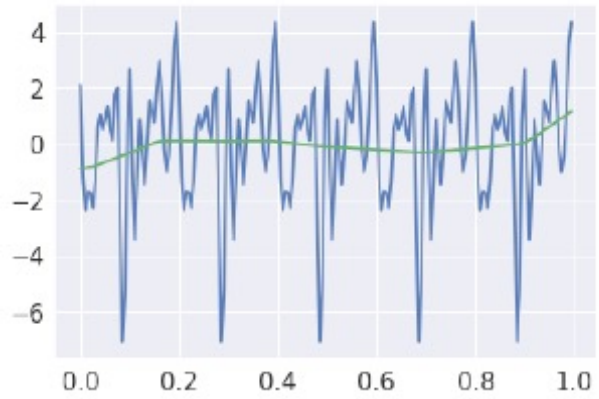
$$\frac{du}{dx} = \cos(\omega x)$$
$$u(0) = 0$$

66,433 free parameters

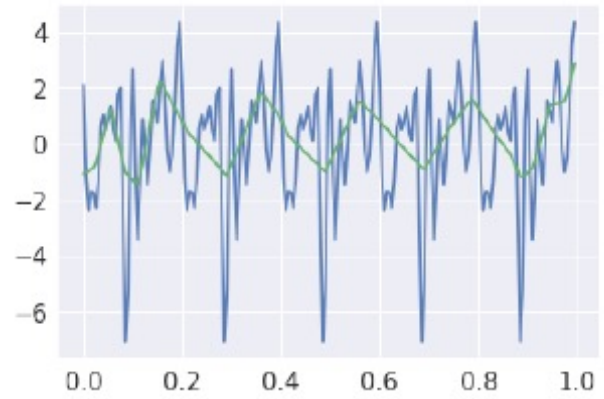
Problem: PINNs **struggle** to solve high-frequency / multiscale problems

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

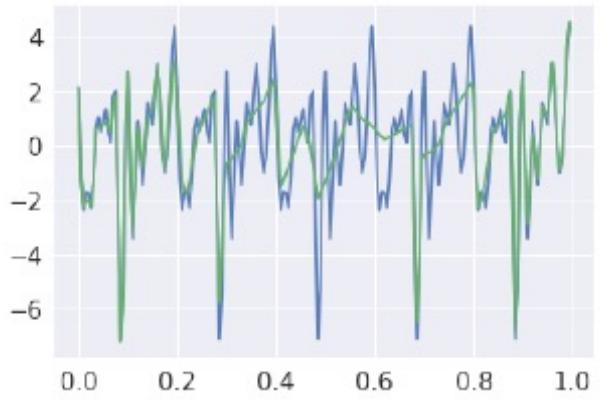
Spectral bias issue



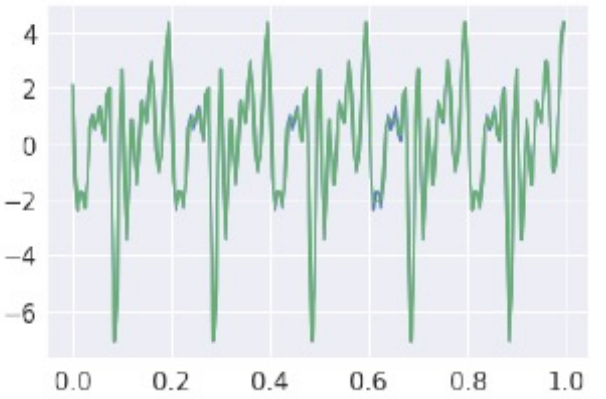
(a) Iteration 100



(b) Iteration 1000



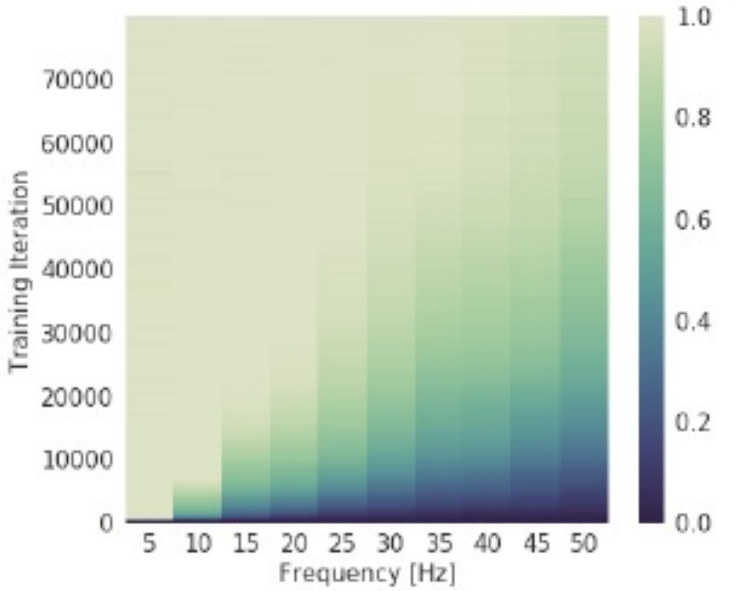
(c) Iteration 10000



(d) Iteration 80000

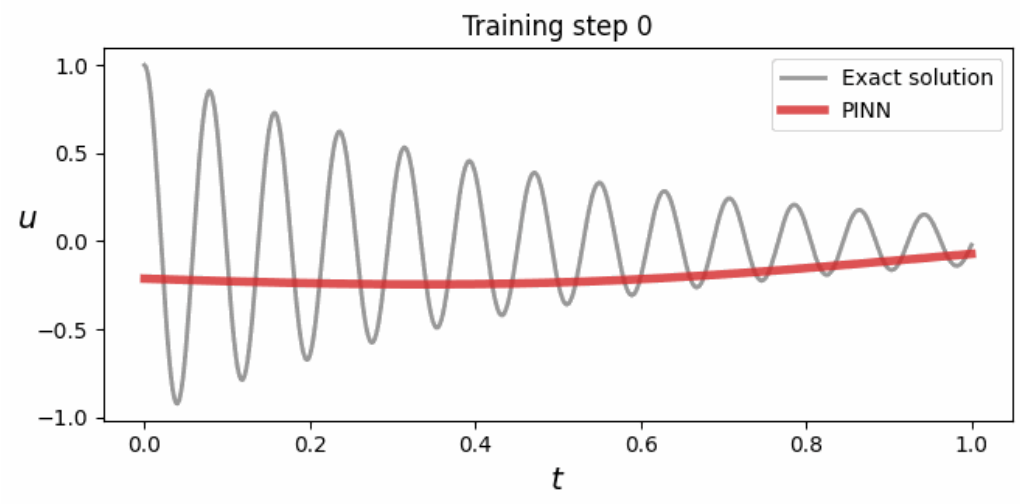
NNs prioritise learning **lower** frequency functions first

Under certain assumptions can be proved via neural tangent kernel theory

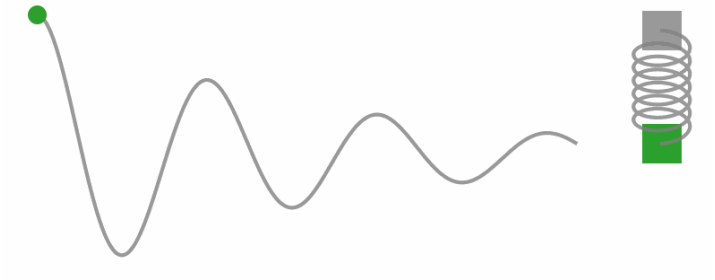


Rahaman, N., et al, On the spectral bias of neural networks. 36th International Conference on Machine Learning, ICML (2019)

Scaling PINNs to higher frequencies



Network size: 2 hidden layers, 64 hidden units



Damped harmonic oscillator

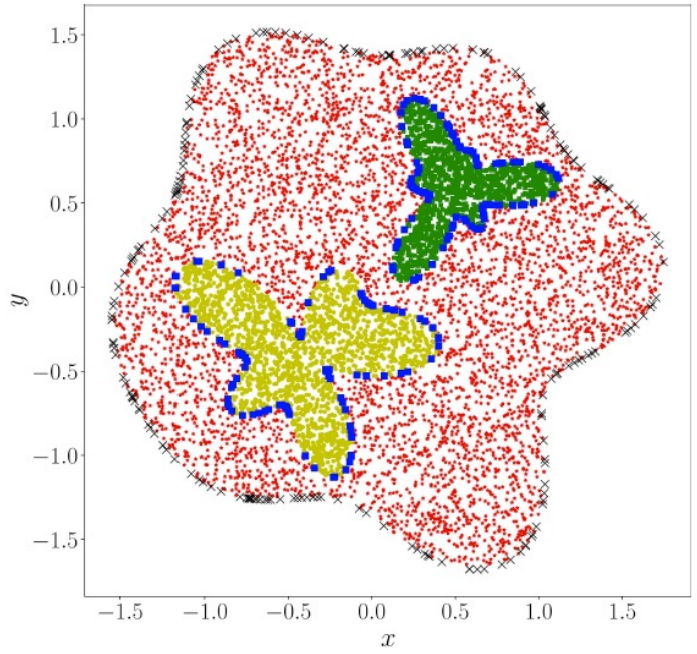
Problem: PINNs **struggle** to solve high-frequency / multiscale problems

As higher frequencies are added:

- More collocation points required
- Larger neural network required
- Spectral bias slows convergence

Leading to a significantly **harder** PINN optimization problem

PINNs + domain decomposition



Jagtap, A., et al., Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. Communications in Computational Physics (2020)

Idea:

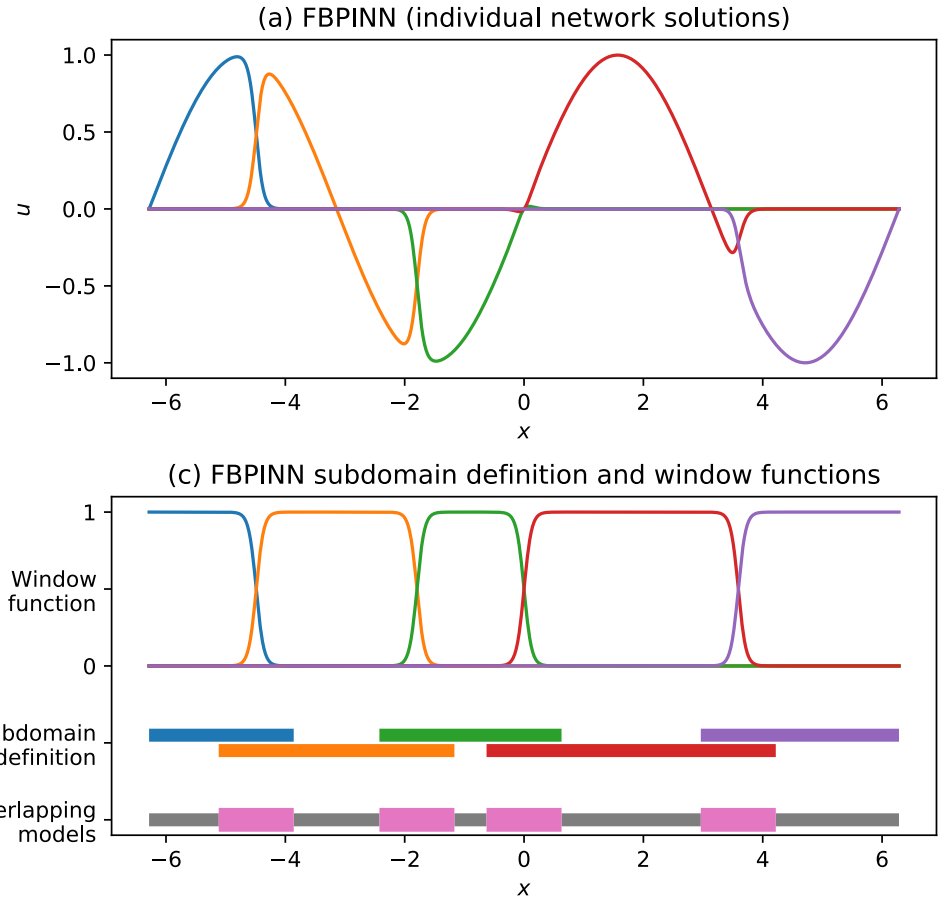
Take a "**divide-and-conquer**" strategy to model more complex problems:

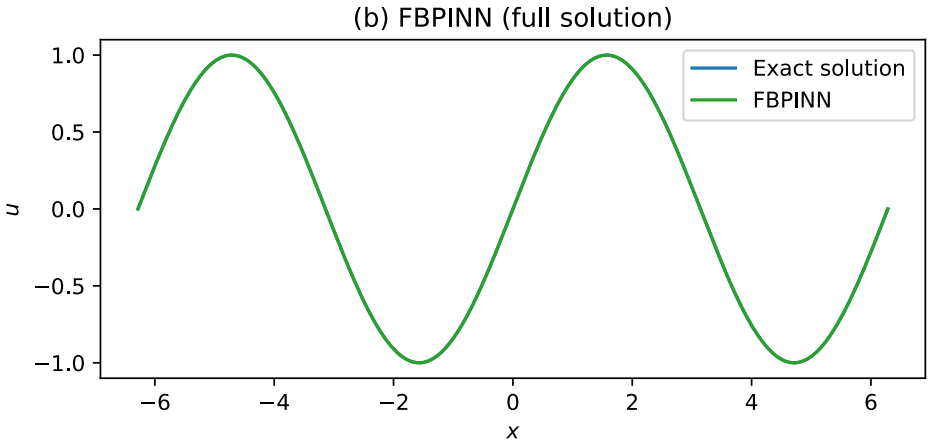
1. Divide modelling domain into many smaller **subdomains**
2. Use a separate neural network in each subdomain to model the solution

Hypothesis:

The resulting (coupled) local optimization problems are easier to solve than a single global problem

Finite basis PINNs (FBPINNs)





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

Window function

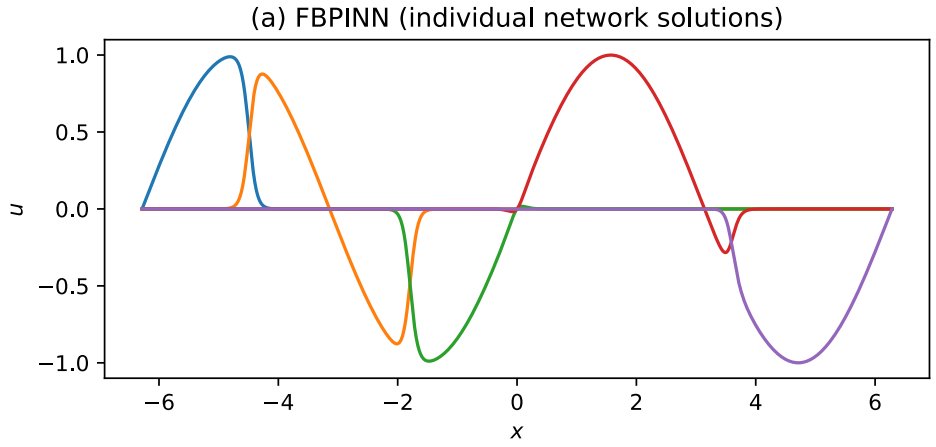
Subdomain network

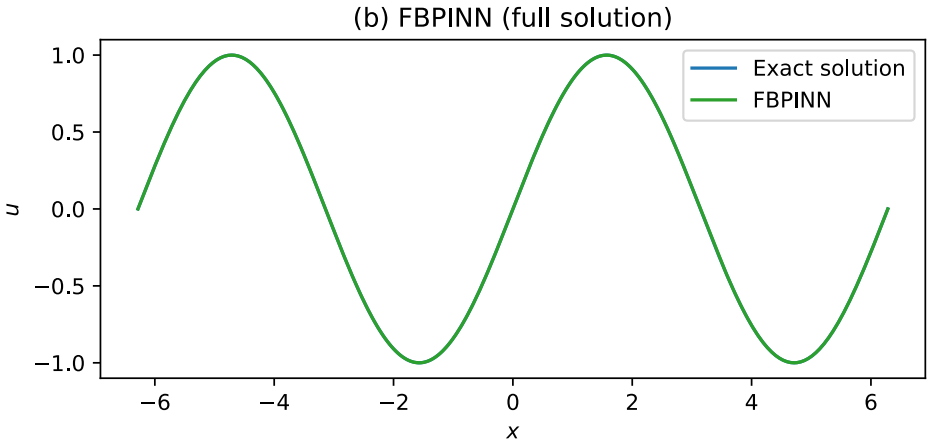
Individual subdomain normalisation

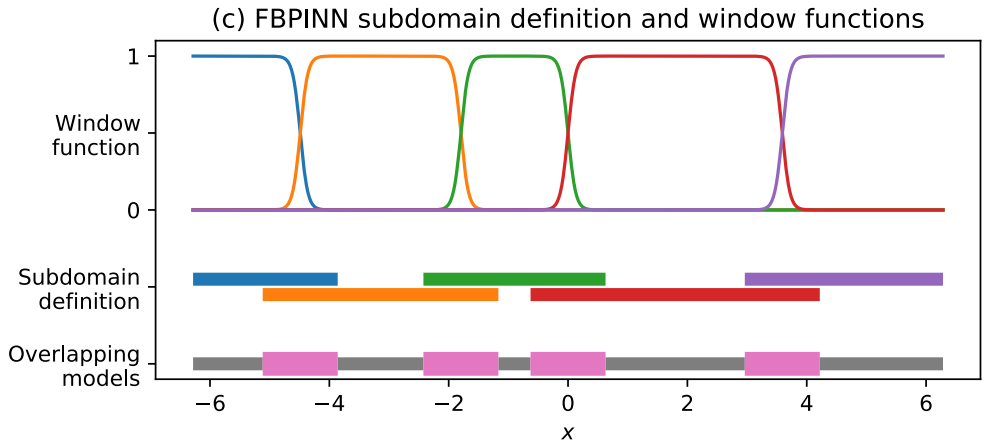
Moseley et al, Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, ACM (2023)

Idea: use **overlapping** subdomains and a **globally** defined solution ansatz

Finite basis PINNs (FBPINNs)





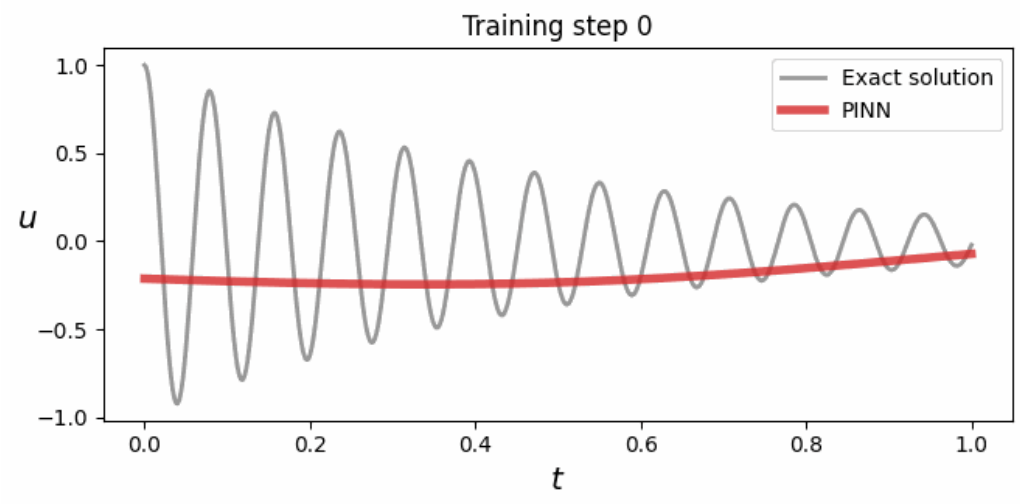


Note:

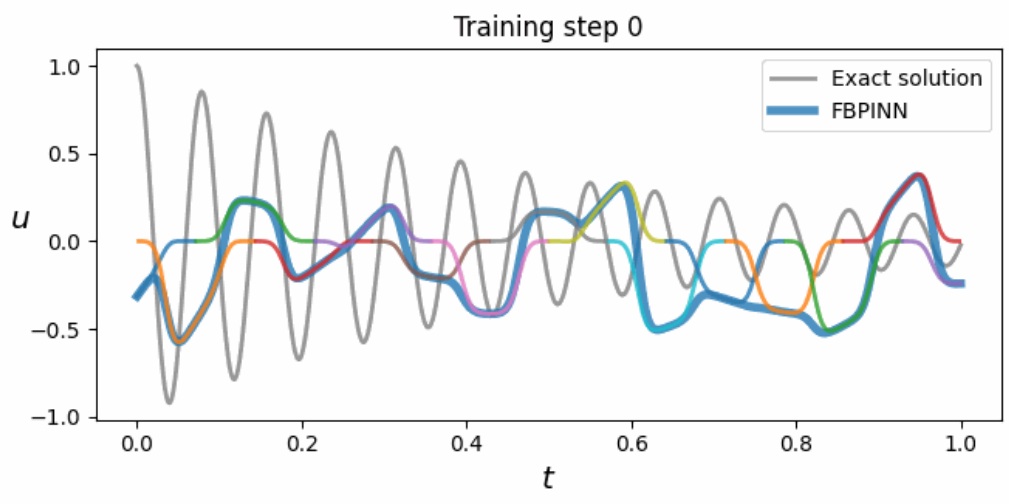
- By construction, FBPINN solution is continuous across subdomain interfaces
- Can be trained with same loss function as PINNs
- FBPINNs can simply be thought of as a custom NN architecture for PINNs

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

FBPINNs vs PINNs

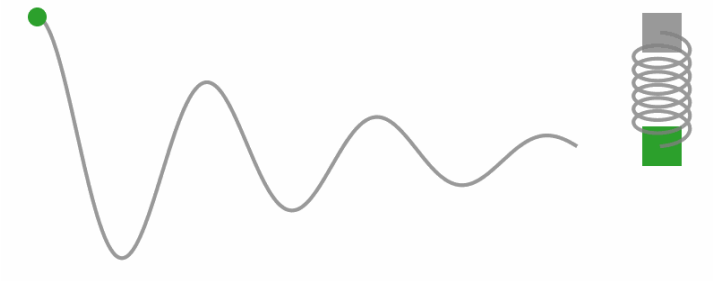


PINN solution



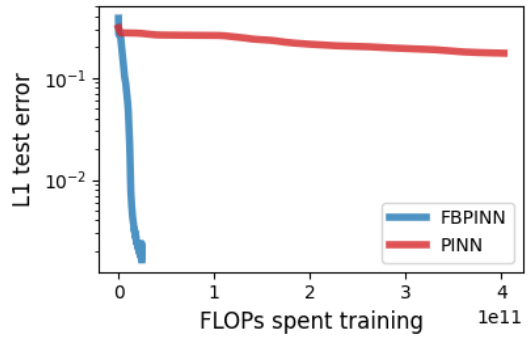
FBPINN solution

Network size: 2 hidden layers, 64 hidden units

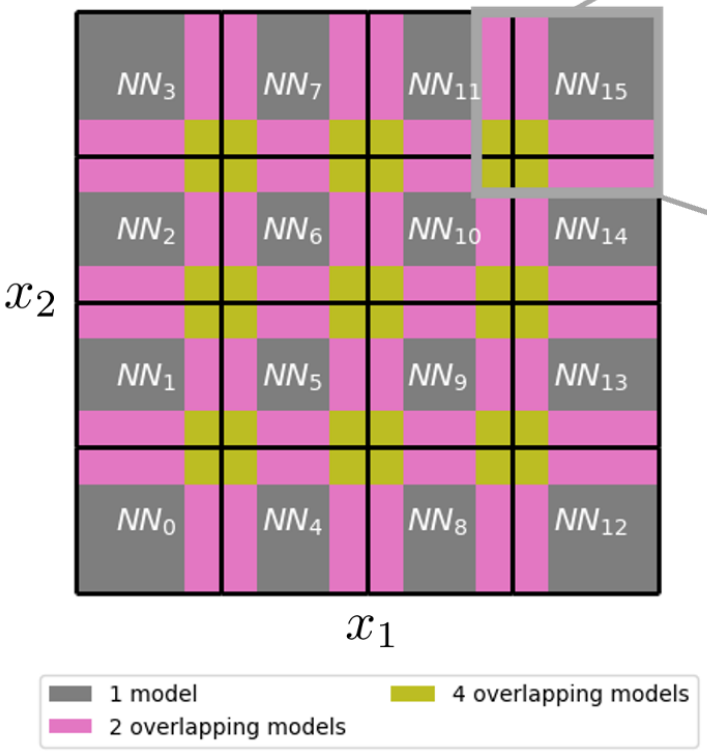


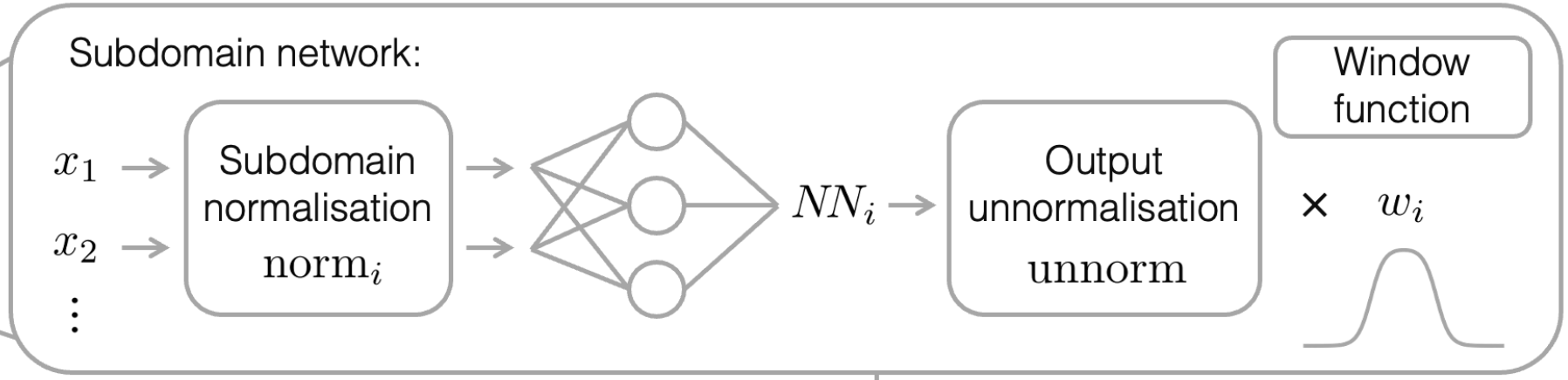
Damped harmonic oscillator

Number of subdomains: 15
Subdomain network size: 1 hidden layer, 32 hidden units



Extension to multiple dimensions





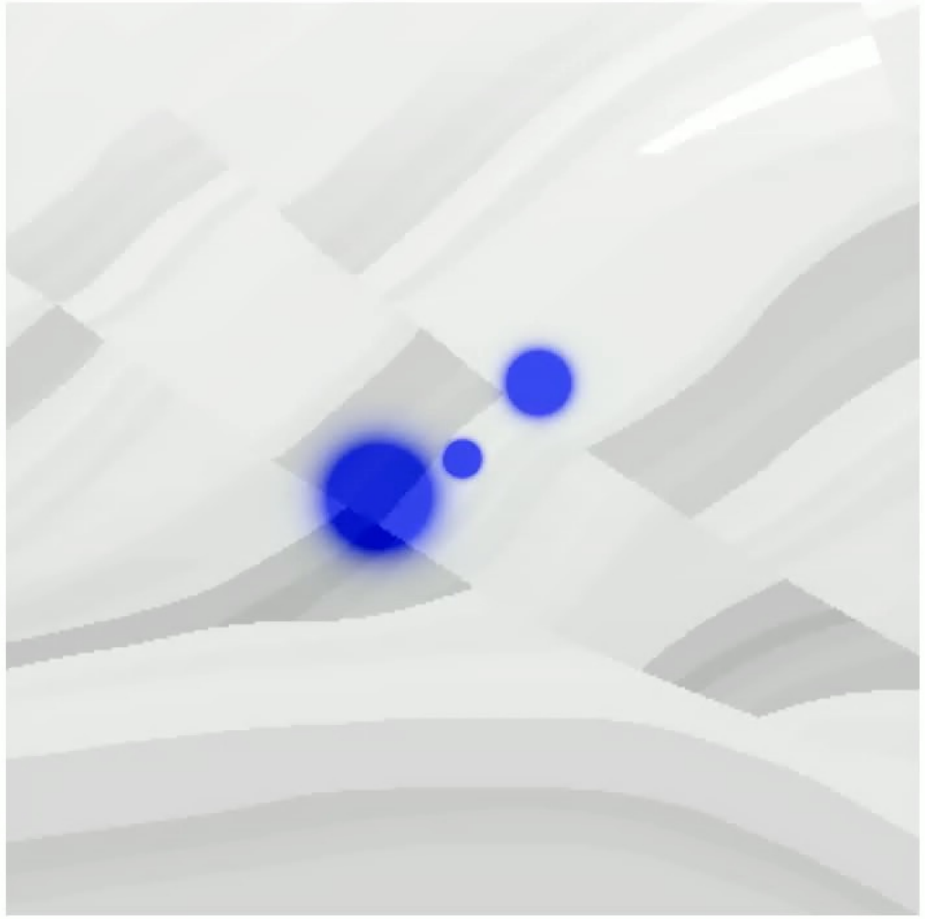
FBPINN solution:

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

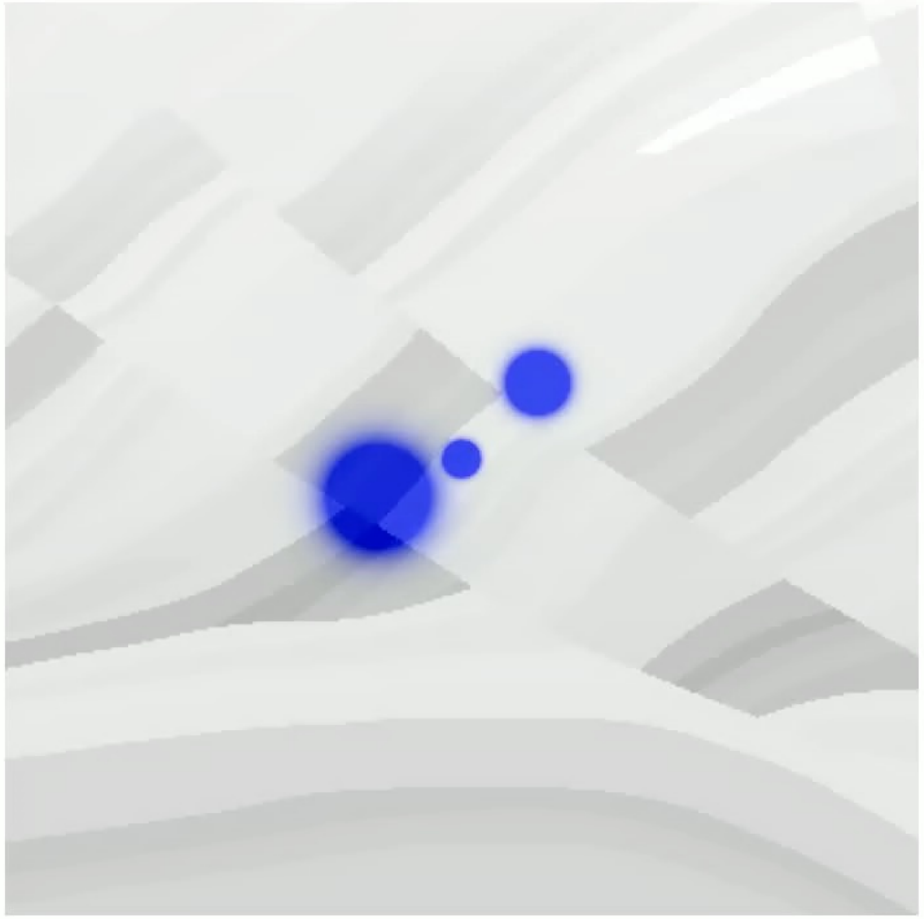
Summation over all subdomain networks

- In general, any overlapping domain decomposition could be used

FBPINN solution



FD simulation



Solving the 2+1D acoustic wave equation:

$$\nabla^2 u(x,t) - \frac{1}{c(x)^2}\frac{\partial^2 u(x,t)}{\partial t^2} = 0$$

Number of subdomains: 60 x 60 x 60 = 216,000
Subdomain network size: 1 hidden layer, 8 hidden units
Total number of free parameters: 9 M
Number of collocation points: 300 x 300 x 300 = 27 M
Optimiser: Time-stepping scheduling, Adam 0.001 lr
Training time: ~5 hr

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

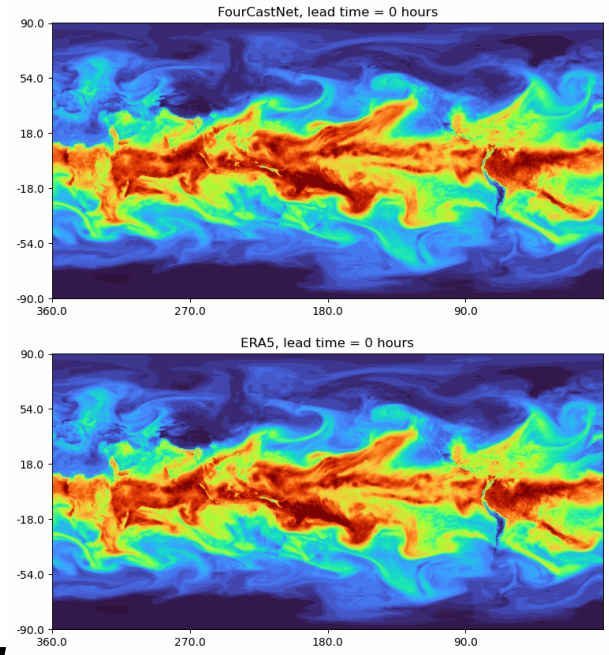
But.. many PINN extensions exist!

PINN limitations and extensions - summary

- Standard 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 which improve these (only some of which were covered above)
- PINNs are an active field of research!

ETH Zurich AI in the Sciences and Engineering Master's Course

AI for science: a revolution?

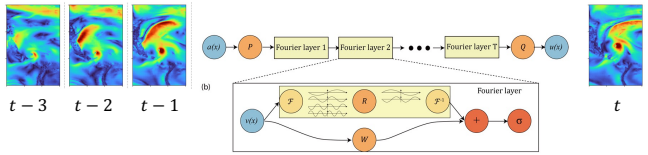


The Washington Post
Democracy Dies in Darkness

WEATHER Extreme Weather Climate Capital Weather Gang Environment Climate Lab

How Big Tech AI models nailed forecast for Hurricane Lee a week in advance

U.S. and European weather agencies are escalating their engagement with artificial intelligence as the technology rapidly advances



Pathak et al, FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, ArXiv (2022)

SCAN ME

65K YouTube views
4.5/5 student feedback score

ETH zürich

YouTube @CAMLabETHZurich

- Aware of advanced **applications** of AI in science
- Understand key scientific machine learning **concepts** and themes