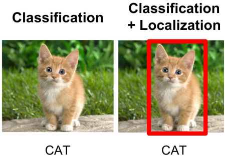
Physics-Informed Neural Network (PINN): Algorithms, Applications, and Software

Lu Lu

Department of Chemical and Biomolecular Engineering
University of Pennsylvania

QCD School
Central China Normal University
Oct 15, 2021

- Computer vision



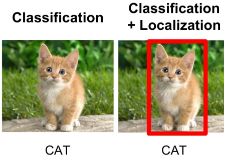
- Machine translation

- Black-box & Trial-and-error
- Big data

Scientific Machine Learning: Learning from Small Data

- *Dinky, Dirty, Dynamic, Deceptive* Data

- Computer vision



- Machine translation

- Black-box & Trial-and-error
- Big data

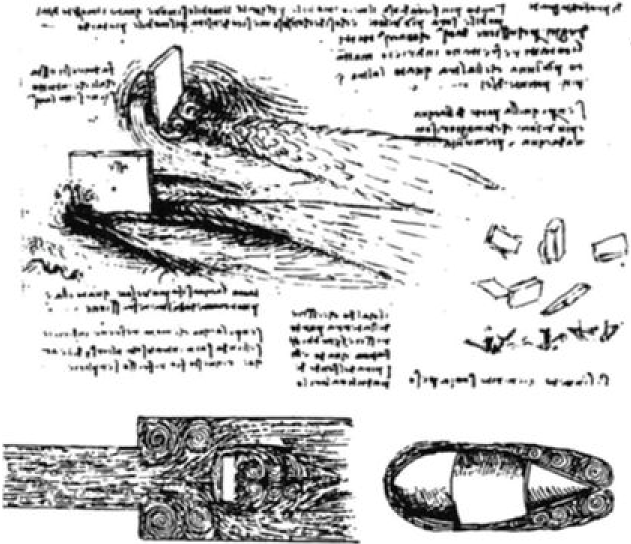
Scientific Machine Learning: Learning from Small Data

- *Dinky, Dirty, Dynamic, Deceptive* Data
- Scientific domain knowledge

  e.g., physical principles, constraints, symmetries, computational simulations
- Accurate, Robust, Reliable, Interpretable, Explainable

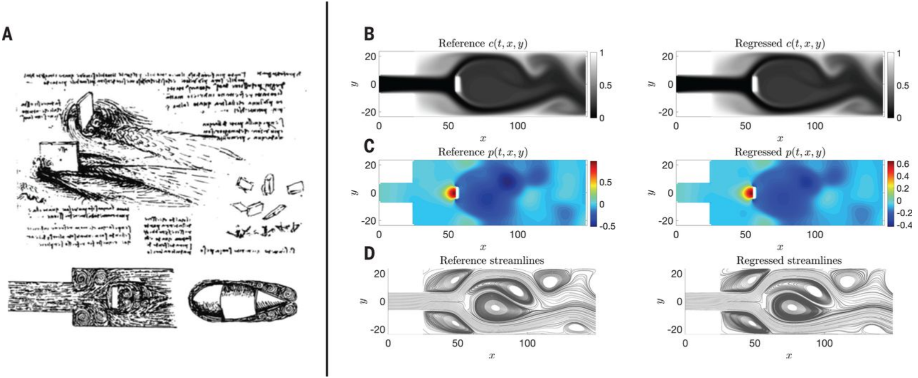
Leonardo da Vinci's drawing of turbulence

"Observe the motion of the surface of the water, Which resembles that of hair, which has two motions ..."



Machine learning with physics

Infer pressure p and velocity fields (u, v) from concentration c

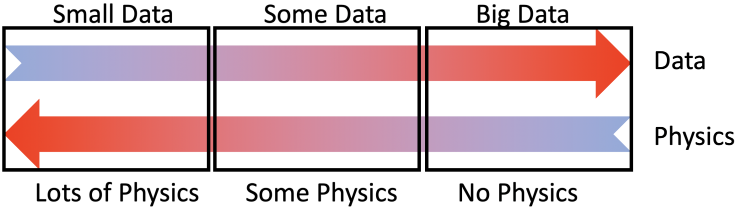


Raissi, Yazdani, & Karniadakis, *Science*, 2020

Data & Physics

Karniadakis, Kevrekidis, **Lu**, et al., *Nature Rev Phys*, 2021

Three scenarios:



1. Lots of physics—Forward problems
    - Finite difference/elements
2. Some physics—Inverse problems
    - Multi-fidelity learning
    - Physics-informed neural network (PINN)
    - DeepM&Mnet
3. No physics—System identification/discovery
    - Operator learning (DeepONet)

Deep learning for partial differential equations (PDEs)

PDE-dependent approaches:

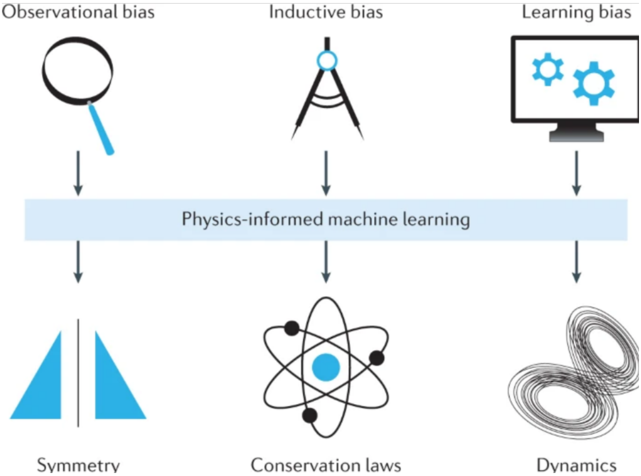
- image-like domain
    - e.g., (Long et al., *ICML*, 2018), (Zhu et al., *J Comput Phys*, 2019)
- parabolic PDEs, e.g., through Feynman-Kac formula
    - e.g., (Beck et al., *J Nonlinear Sci*, 2017), (Han et al., *PNAS*, 2018)
- variational form
    - e.g., (E & Yu, *Commun Math Stat*, 2018)

General approaches:

- Galerkin type projection
    - e.g., (Meade & Fernandez, *Math Comput Model*, 1994), (Kharazmi et al., *arXiv*, 2019)
- strong form (Physics-informed neural networks)
    - e.g., (Dissanayake & Phan-Thien, *Commun Numer Meth En*, 1994), (Lagaris et al., *IEEE Trans Neural Netw*, 1998), (Raissi et al., *J Comput Phys*, 2019)

How to embed physics in ML?

Principles of physics-informed learning:



Karniadakis, Kevrekidis, **Lu**, et al., *Nature Rev Phys*, 2021

Inverse problems

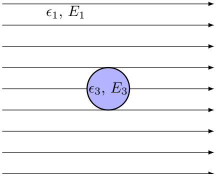
Challenge: *small* data + *incomplete* physics laws

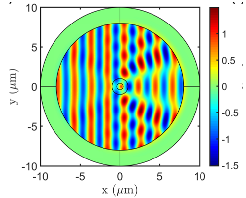
Invisible cloaking

joint work with Prof. Luca Dal Negro (Boston U)

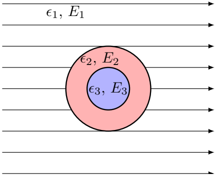
Permittivity ϵ, permeability μ

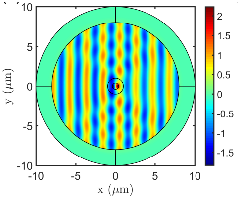
Electric field E without coating





Electric field E with coating

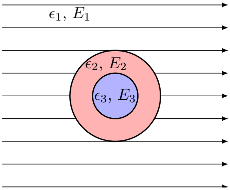




Chen, **Lu**, et al., *Opt Express*, 2020

Invisible cloaking

Goal: Given ϵ_1 and ϵ_3, find $\epsilon_2(x,y)$ s.t. $E_1 \approx E_{1,target}$



Helmholtz equation $(k_0 = \frac{2\pi}{\lambda_0})$

$$\nabla^2 E_i + \epsilon_i k_0^2 E_i = 0, \qquad i = 1,2,3$$

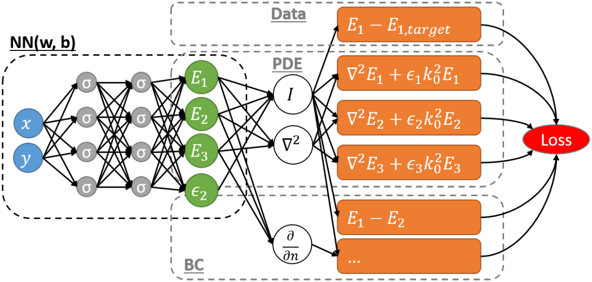
Boundary conditions:

- Outer circle: $E_1 = E_2,\ \frac{1}{\mu_1}\frac{\partial E_1}{\partial \mathbf{n}} = \frac{1}{\mu_2}\frac{\partial E_2}{\partial \mathbf{n}}$
- Inner circle: $E_2 = E_3,\ \frac{1}{\mu_2}\frac{\partial E_2}{\partial \mathbf{n}} = \frac{1}{\mu_3}\frac{\partial E_3}{\partial \mathbf{n}}$

Chen, **Lu**, et al., *Opt Express*, 2020

Physics-informed neural networks (PINNs)

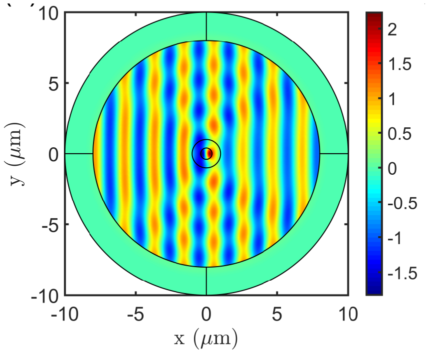
Idea: Embed a PDE into the loss via automatic differentiation (AD)



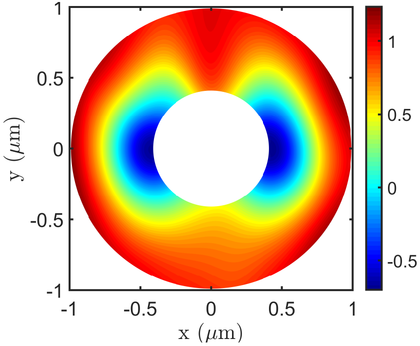
- mesh-free & particle-free
- inverse problems: seamlessly integrate data and physics
- black-box or noisy IC/BC/forcing terms (Pang*, **Lu***, et al., *SIAM J Sci Comput*, 2019)
- a unified framework: PDE, integro-differential equations (**Lu** et al., *SIAM Rev*, 2021), fractional PDE (Pang*, **Lu***, et al., *SIAM J Sci Comput*, 2019), stochastic PDE (Zhang, **Lu**, et al., *J Comput Phys*, 2019)

Invisible cloaking

Electric field E_i



Permittivity ϵ_2



Chen, **Lu**, et al., *Opt Express*, 2020

Physics-informed neural networks (PINNs)

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \dots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} \in \Omega$$

- Initial/boundary conditions $\mathcal{B}(u, \mathbf{x}) = 0 \quad \text{on} \quad \partial\Omega$
- Extra information $\mathcal{I}(u, \mathbf{x}) = 0 \quad \text{for} \quad \mathbf{x} \in \mathcal{T}_i$

$$\min_{\boldsymbol{\theta}, \boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_b) + w_i \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i)$$

where

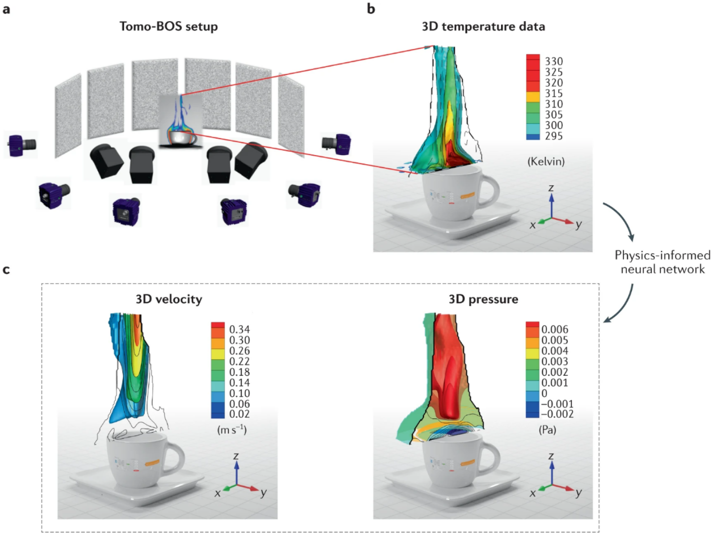
$$\mathcal{L}_f = \frac{1}{|\mathcal{T}_f|} \sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots; \boldsymbol{\lambda}\right) \right\|_2^2$$

$$\mathcal{L}_b = \frac{1}{|\mathcal{T}_b|} \sum_{\mathbf{x} \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2$$

$$\mathcal{L}_i = \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{x} \in \mathcal{T}_i} \|\mathcal{I}(\hat{u}, \mathbf{x})\|_2^2$$

Inferring the flow over an espresso cup

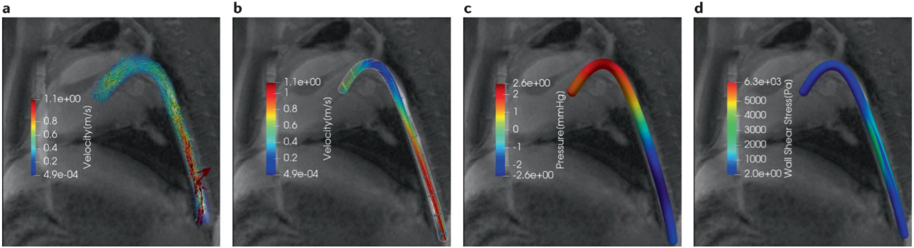
Data: A video of temperature field.



Cai et al., *J Fluid Mech*, 2021

Filtering of *in-vivo* 4D-flow magnetic resonance imaging (MRI) data of blood flow in a porcine descending aorta

- MRI data: Very coarse resolution & heavily corrupted by noise



- Reconstructions of the velocity and pressure fields
- Identify regions of no-slip flow, from which one can reconstruct the location and motion of the arterial wall

---

Wang, Kissas, & Perdikaris

Error analysis

Theorem (Universal approximation theorem; Cybenko, 1989)

Let σ be any continuous sigmoidal function. Then finite sums of the form $G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j \cdot x + b_j)$ are dense in $C(I_d)$.

Theorem (Pinkus, 1999)

Let $\mathbf{m}^i \in \mathbb{Z}_+^d$, $i = 1, \ldots, s$, and set $m = \max_{i=1,\ldots,s}(m_1^i + \cdots + m_d^i)$. Assume $\sigma \in C^m(\mathbb{R})$ and σ is not a polynomial. Then the space of single hidden layer neural nets

$$\mathcal{M}(\sigma) := span\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

is dense in

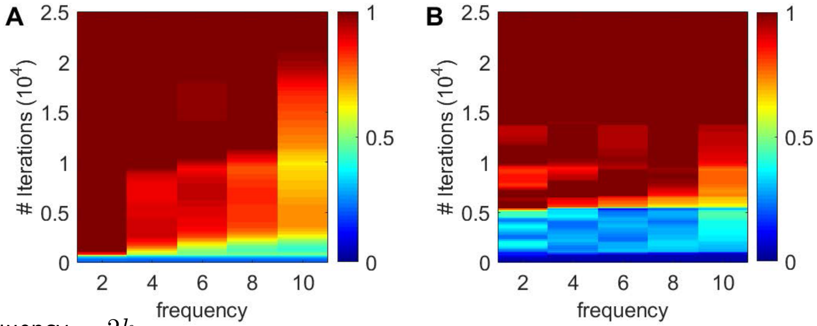
$$C^{\mathbf{m}^1,\ldots,\mathbf{m}^s}(\mathbb{R}^d) := \cap_{i=1}^{s} C^{\mathbf{m}^i}(\mathbb{R}^d).$$

Optimization & Generalization:

Shin et al., 2020; Mishra & Molinaro, 2020; Luo & Yang, 2020

Optimization

- A: approximate $f(x) = \sum_{k=1}^{5} \sin(2kx)/(2k)$
  - learn from low to high frequencies (Rahaman et al., *ICML*, 2019; Xu et al., arXiv, 2019)
- B: solve the Poisson equation $-f_{xx} = \sum_{k=1}^{5} 2k \sin(2kx)$
  - all frequencies are learned almost simultaneously
  - faster learning



Frequency $= 2k$

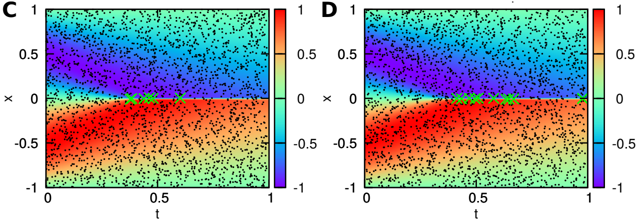
Lu et al., *SIAM Rev*, 2021

Generalization: Residual-based adaptive refinement (RAR)

Challenge: Uniform residual points are not efficient for PDEs with steep solutions.
e.g., Burgers equation ($x \in [-1, 1],\ t \in [0, 1]$):

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad u(x, 0) = -\sin(\pi x), \quad u(-1, t) = u(1, t) = 0.$$

- **Idea**: adaptively add more points in locations with large PDE residual $\left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda}\right)\right\|$



10,000 (Raissi et al., *J Comput Phys*, 2019) ↓ 2,540

Lu et al., *SIAM Rev*, 2021

Hard constraints in inverse design/topology optimization

Constrained optimization problem for $\gamma(\mathbf{x})$

$$\min_{\mathbf{u},\gamma} \mathcal{J}(\mathbf{u};\gamma)$$

subject to

$$\begin{cases} \mathcal{F}[\mathbf{u};\gamma] = \mathbf{0}, \\ \mathcal{B}[\mathbf{u}] = 0, \\ h(\mathbf{u},\gamma) \leq 0, \end{cases}$$

Convert the constrained optimization to an unconstrained optimization via losses:

$$\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \frac{1}{MN} \sum_{j=1}^{M} \sum_{i=1}^{N} \left| \mathcal{F}_i \left[\hat{\mathbf{u}}(\mathbf{x}_j); \hat{\gamma}(\mathbf{x}_j) \right] \right|^2,$$

$$\mathcal{L}_h(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \mathbb{1}_{\{h(\hat{\mathbf{u}},\hat{\gamma})>0\}} h^2(\hat{\mathbf{u}}, \hat{\gamma}),$$

Hard constraints

- Soft constraints: large $\mu_{\mathcal{F}}, \mu_h \Rightarrow$ ill-conditioned optimization

$$\mathcal{L}(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \mathcal{J} + \mu_{\mathcal{F}}\mathcal{L}_{\mathcal{F}} + \mu_h\mathcal{L}_h$$

- Penalty method (a sequence of soft constraints)

$$\mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \mathcal{J} + \mu_{\mathcal{F}}^k\mathcal{L}_{\mathcal{F}} + \mu_h^k\mathcal{L}_h$$

$$\mu_{\mathcal{F}}^{k+1} = \beta_{\mathcal{F}}\mu_{\mathcal{F}}^k, \quad \mu_h^{k+1} = \beta_h\mu_h^k$$

- Augmented Lagrangian method: new terms to mimic Lagrange multipliers

$$\begin{aligned}\mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \quad & \mathcal{J} + \mu_{\mathcal{F}}^k\mathcal{L}_{\mathcal{F}} + \mu_h^k\mathbb{1}_{\{h>0 \vee \lambda_h^k>0\}}h^2 \\ & + \frac{1}{MN}\sum_{j=1}^M\sum_{i=1}^N \lambda_{i,j}^k\mathcal{F}_i\left[\hat{\mathbf{u}}(\mathbf{x}_j); \hat{\gamma}(\mathbf{x}_j)\right] + \lambda_h^k h\end{aligned}$$

$$\lambda_{i,j}^k = \lambda_{i,j}^{k-1} + 2\mu_{\mathcal{F}}^{k-1}\mathcal{F}_i\left[\hat{\mathbf{u}}(\mathbf{x}_j; \boldsymbol{\theta}_u^{k-1}); \hat{\gamma}(\mathbf{x}_j; \boldsymbol{\theta}_\gamma^{k-1})\right]$$

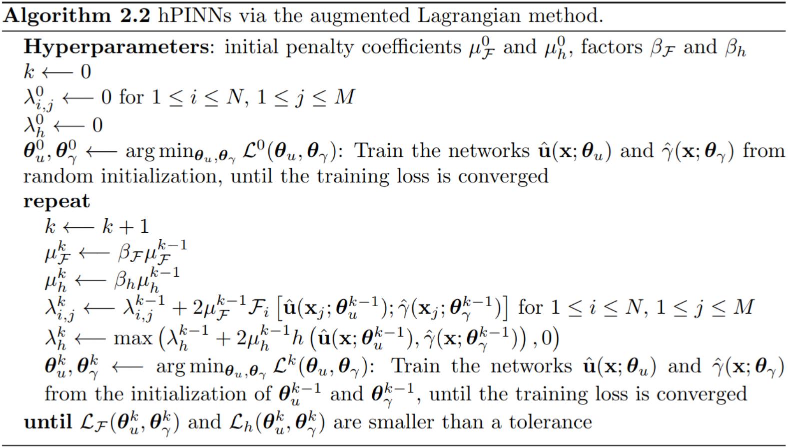
$$\lambda_h^k = \max\left(\lambda_h^{k-1} + 2\mu_h^{k-1}h\left(\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u^{k-1}), \hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma^{k-1})\right), 0\right)$$

Lu et al., arXiv:2102.04626

hPINN: PINN with hard constraints

Augmented Lagrangian method

$$\begin{aligned}\mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \quad & \mathcal{J} + \mu_\mathcal{F}^k \mathcal{L}_\mathcal{F} + \mu_h^k \mathbb{1}_{\{h>0 \vee \lambda_h^k>0\}} h^2 \\ & + \frac{1}{MN} \sum_{j=1}^{M} \sum_{i=1}^{N} \lambda_{i,j}^k \mathcal{F}_i \left[\hat{\mathbf{u}}(\mathbf{x}_j); \hat{\gamma}(\mathbf{x}_j)\right] + \lambda_h^k h\end{aligned}$$



Algorithm 2.2 hPINNs via the augmented Lagrangian method.

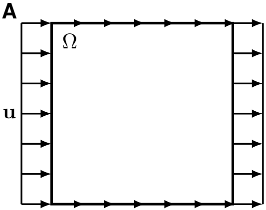
```
Hyperparameters: initial penalty coefficients $\mu_\mathcal{F}^0$ and $\mu_h^0$, factors $\beta_\mathcal{F}$ and $\beta_h$
$k \longleftarrow 0$
$\lambda_{i,j}^0 \longleftarrow 0$ for $1 \le i \le N$, $1 \le j \le M$
$\lambda_h^0 \longleftarrow 0$
$\boldsymbol{\theta}_u^0, \boldsymbol{\theta}_\gamma^0 \longleftarrow \arg\min_{\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma} \mathcal{L}^0(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma)$: Train the networks $\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u)$ and $\hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma)$ from random initialization, until the training loss is converged
repeat
    $k \longleftarrow k + 1$
    $\mu_\mathcal{F}^k \longleftarrow \beta_\mathcal{F} \mu_\mathcal{F}^{k-1}$
    $\mu_h^k \longleftarrow \beta_h \mu_h^{k-1}$
    $\lambda_{i,j}^k \longleftarrow \lambda_{i,j}^{k-1} + 2\mu_\mathcal{F}^{k-1} \mathcal{F}_i \left[\hat{\mathbf{u}}(\mathbf{x}_j; \boldsymbol{\theta}_u^{k-1}); \hat{\gamma}(\mathbf{x}_j; \boldsymbol{\theta}_\gamma^{k-1})\right]$ for $1 \le i \le N$, $1 \le j \le M$
    $\lambda_h^k \longleftarrow \max\left(\lambda_h^{k-1} + 2\mu_h^{k-1} h\left(\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u^{k-1}), \hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma^{k-1})\right), 0\right)$
    $\boldsymbol{\theta}_u^k, \boldsymbol{\theta}_\gamma^k \longleftarrow \arg\min_{\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma} \mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma)$: Train the networks $\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u)$ and $\hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma)$ from the initialization of $\boldsymbol{\theta}_u^{k-1}$ and $\boldsymbol{\theta}_\gamma^{k-1}$, until the training loss is converged
until $\mathcal{L}_\mathcal{F}(\boldsymbol{\theta}_u^k, \boldsymbol{\theta}_\gamma^k)$ and $\mathcal{L}_h(\boldsymbol{\theta}_u^k, \boldsymbol{\theta}_\gamma^k)$ are smaller than a tolerance
```

Lu et al., arXiv:2102.04626

Topology optimization of fluids in Stokes flow

- Solid: $\rho = 0$; fluid: $\rho = 1$
- Minimize a objective of dissipated power

$$\mathcal{J} = \int_\Omega \left(\frac{1}{2} \nabla \mathbf{u} : \nabla \mathbf{u} + \frac{1}{2} \alpha \mathbf{u}^2 \right) dxdy$$



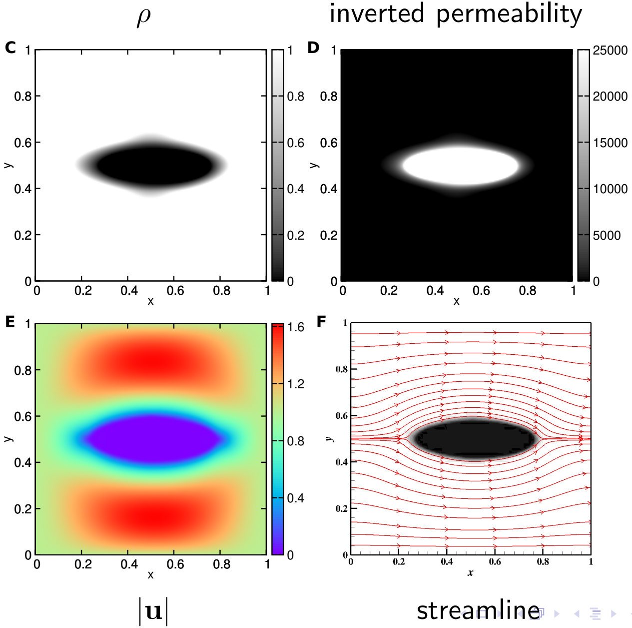
- Generalized Stokes equation with Darcy's law

$$-\nu \Delta \mathbf{u} + \nabla p = \mathbf{f}$$
$$\nabla \cdot \mathbf{u} = 0$$

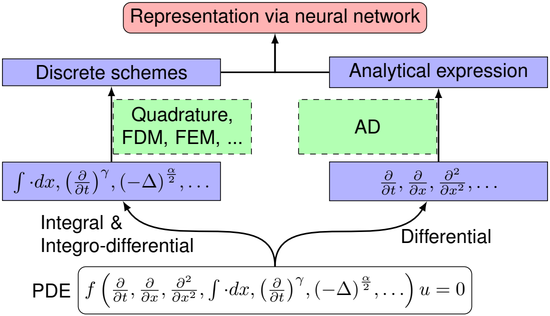
  - $\mathbf{f} = \alpha \mathbf{u}$: the Brinkman term
  - α: inverted permeability $\alpha(\rho) = \overline{\alpha} + (\underline{\alpha} - \overline{\alpha}) \rho \frac{1+q}{\rho+q}$
  - fluid volume constraint:

$$\int_\Omega \rho \, dxdy \leq \gamma = 0.9$$

Topology optimization of fluids in Stokes flow



PINNs for solving integro-differential equations



Representation via neural network

Discrete schemes

Analytical expression

Quadrature, FDM, FEM, ...

AD

$\int \cdot dx, \left(\frac{\partial}{\partial t}\right)^{\gamma}, (-\Delta)^{\frac{\alpha}{2}}, \ldots$

$\frac{\partial}{\partial t}, \frac{\partial}{\partial x}, \frac{\partial^2}{\partial x^2}, \ldots$

Integral & Integro-differential

Differential

PDE $\boxed{f\left(\frac{\partial}{\partial t}, \frac{\partial}{\partial x}, \frac{\partial^2}{\partial x^2}, \int \cdot dx, \left(\frac{\partial}{\partial t}\right)^{\gamma}, (-\Delta)^{\frac{\alpha}{2}}, \ldots\right) u = 0}$

e.g., Riemann-Liouville directional fractional derivative of order $\alpha \in (1, 2)$

$$D_{\boldsymbol{\theta}}^{\alpha} u(\boldsymbol{x}) = \frac{1}{\Gamma(2-\alpha)} (\boldsymbol{\theta} \cdot \nabla)^2 \int_0^{+\infty} \xi^{1-\alpha} u(\boldsymbol{x} - \xi\boldsymbol{\theta}) d\xi \qquad \boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^D$$

$$= \frac{1}{(\Delta x)^{\alpha}} \sum_{k=1}^{\lceil \lambda d(\boldsymbol{x}, \boldsymbol{\theta}, \Omega) \rceil} (-1)^k \binom{\alpha}{k} u(\boldsymbol{x} - (k-1)\Delta x \boldsymbol{\theta}) + \mathcal{O}(\Delta x)$$

Pang*, **Lu***, et al., *SIAM J Sci Comput*, 2019

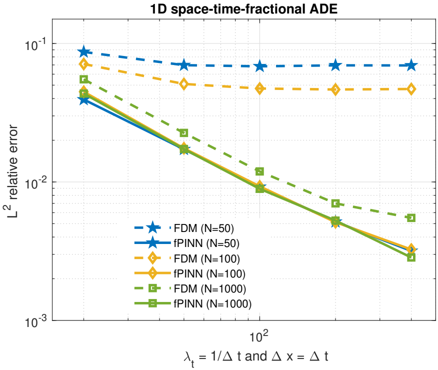
Lu et al., *SIAM Rev*, 2021

Space-time-fractional PDEs with black-box (BB) terms

Fractional advection-diffusion equation with zero BC

$$\frac{\partial^\gamma u(\boldsymbol{x},t)}{\partial t^\gamma} = -c(-\Delta)^{\alpha/2}u(\boldsymbol{x},t) - \boldsymbol{v}\cdot\nabla u(\boldsymbol{x},t) + f_{BB}(\boldsymbol{x},t)$$

Forcing term f_{BB} only known at scattered spatio-temporal points



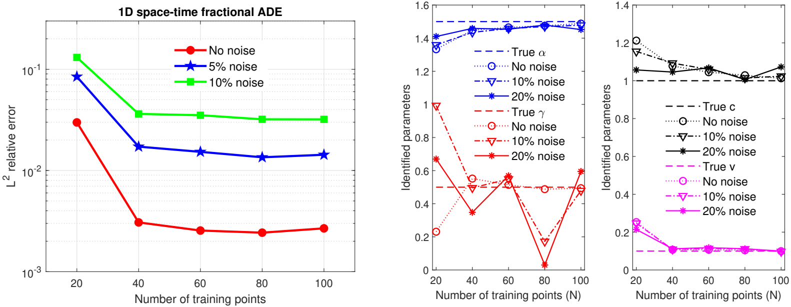
PINN vs. FDM: no interpolation error

Pang*, **Lu***, et al., *SIAM J Sci Comput*, 2019

Space-time-fractional PDEs with noisy terms

$$\frac{\partial^\gamma u(\boldsymbol{x},t)}{\partial t^\gamma} = -c(-\Delta)^{\alpha/2}u(\boldsymbol{x},t) - \boldsymbol{v}\cdot\nabla u(\boldsymbol{x},t) + f_{BB}(\boldsymbol{x},t)$$

Gaussian noise added to the forcing term f_{BB}
(left) Forward problem & (right) Inverse problem



PINN: noise easily handled by regularization

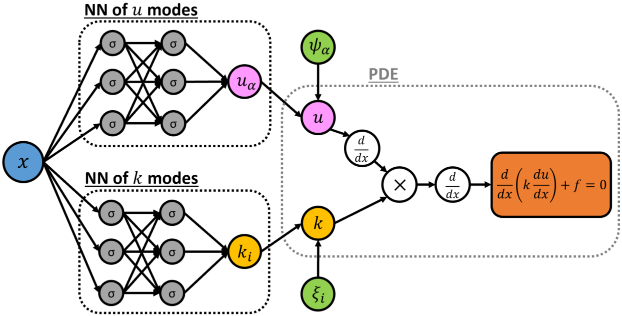
Pang*, **Lu***, et al., *SIAM J Sci Comput*, 2019

PINNs for solving stochastic PDEs

$$-\frac{d}{dx}\left(k(x;\omega)\frac{du}{dx}\right) = f(x), \quad x \in [-1,1], \quad \omega \in \Omega$$

$$u(-1) = u(1) = 0$$

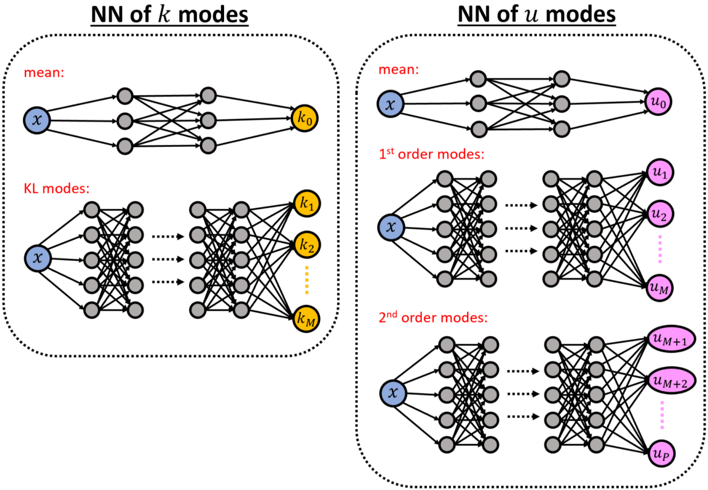
- Karhunen-Loève expansion: $k(x;\omega_s) = \widehat{k_0}(x) + \sum_{i=1}^{M} \sqrt{\lambda_i}\widehat{k_i}(x)\xi_{s,i}$
- arbitrary polynomial chaos: $u(x;\omega_s) = \sum_{\alpha=0}^{P} \widehat{u_\alpha}(x)\psi_\alpha(\boldsymbol{\xi_s})$



Zhang, **Lu**, et al., *J Comput Phys*, 2019

PINNs for solving stochastic PDEs

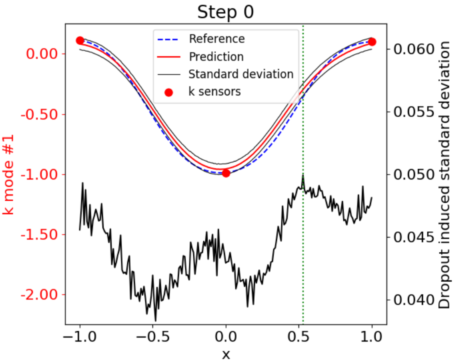
Use different NNs to learn the mean and modes of different scales

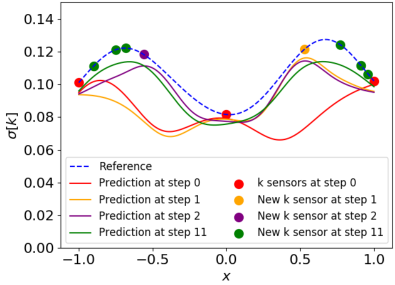


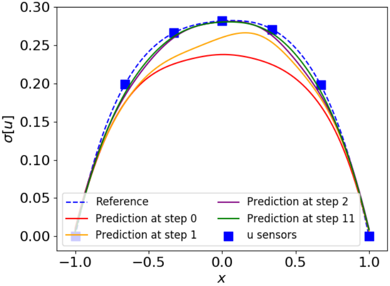
Zhang, **Lu**, et al., *J Comput Phys*, 2019

Active learning for stochastic inverse problem

Dropout-induced uncertainty serves as the guidance for *active learning*.







Zhang, **Lu**, et al., *J Comput Phys*, 2019

Open-source software: **DeepXDE**

nature

NEWS FEATURE · 20 JANUARY 2021

Ten computer codes that transformed science

From Fortran to arXiv.org, these advances in programming and platforms sent biology, climate science and physics into warp speed.

Search projects

DeepXDE 0.13.6

Physics-informed deep learning

> 100,000 downloads, 600 GitHub Stars

BROWN

University of Victoria

PUC RIO

Los Alamos NATIONAL LABORATORY

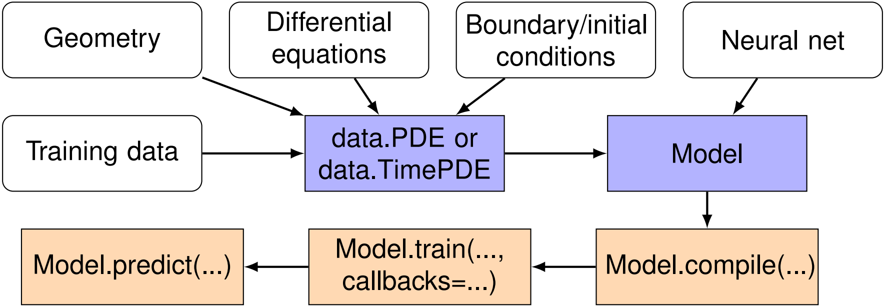
Sandia National Laboratories

Ansys

SIEMENS

Usage of DeepXDE

Solving differential equations in DeepXDE is no more than **specifying the problem using the build-in modules**.



Geometry

Differential equations

Boundary/initial conditions

Neural net

Training data

data.PDE or data.TimePDE

Model

Model.predict(...)

Model.train(..., callbacks=...)

Model.compile(...)

I. Time-independent problems: Poisson equation

2D Poisson equation over an L-shaped domain $\Omega = [-1,1]^2 \setminus [0,1]^2$:

$$-\Delta u(x,y) = 1, \quad (x,y) \in \Omega, \qquad u(x,y) = 0, \quad (x,y) \in \partial\Omega$$

1. geometry

```python
geom = dde.geometry.Polygon(
    [[0, 0], [1, 0], [1, -1], [-1, -1], [-1, 1], [0, 1]])
```

2. PDE via automatic differentiation

```python
def pde(x, y):
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    dy_yy = dde.grad.hessian(y, x, i=1, j=1)
    return -dy_xx - dy_yy - 1
```

3. BC: Dirichlet, Neumann, Robin, periodic, and a general BC

```python
def boundary(x, on_boundary):
    return on_boundary  # Default: entire geometry boundary
def func(x):
    return 0  # Value

bc = dde.DirichletBC(geom, func, boundary)
```

I. Time-independent problems: Poisson equation

4. “data”: geometry + PDE + BC + “training” points

```python
data = dde.data.PDE(
    geom, pde, bc, num_domain=1000, num_boundary=100, ...)
```

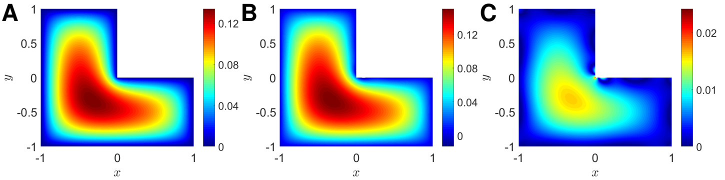
5. network, e.g., feed-forward network

```python
net = dde.maps.FNN([2]+[50]*4+[1], "tanh", "Glorot uniform",
    ...)
```

6. model: data + network, and train

```python
model = dde.Model(data, net)
model.compile("adam", lr=0.001, ...)
model.train(epochs=50000, ...)
```

(A) spectral element, (B) PINN, (C) error



II. Time-dependent problems: Diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 x}{\partial x^2} - e^{-t}(1-\pi^2)\sin(\pi x), \quad x \in [-1,1], t \in [0,1]$$

with Dirichlet BC. (Exact solution $u(x,t) = e^{-t}\sin(\pi x)$)

- geometry

```
geom = dde.geometry.Interval(-1, 1)
timedomain = dde.geometry.TimeDomain(0, 1)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

- IC, similar to Dirichlet BC

```
def func(x):
    return np.sin(np.pi * x[:, 0:1]) * np.exp(-x[:, 1:])

ic = dde.IC(geomtime, func, lambda _, on_initial: on_initial)
```

- “data”: geometry + PDE + BC/IC + “training” points

```
data = dde.data.TimePDE(
    geomtime, ..., [bc, ic],
    num_domain=40, num_boundary=20, num_initial=10, ...)
```

III. ODE/PDE system: Lorenz system

$$\frac{dx}{dt} = \rho(y - x), \quad \frac{dy}{dt} = x(\sigma - z) - y, \quad \frac{dz}{dt} = xy - \beta z$$

- ODE system

```
def Lorenz_system(x, y):
    y1, y2, y3 = y[:, 0:1], y[:, 1:2], y[:, 2:]
    dy1_x = dde.grad.jacobian(y, x, i=0)
    dy2_x = dde.grad.jacobian(y, x, i=1)
    dy3_x = dde.grad.jacobian(y, x, i=2)
    return [
        dy1_x - C1 * (y2 - y1),
        dy2_x - y1 * (C2 - y3) + y2,
        dy3_x - y1 * y2 + C3 * y3
    ]
```

- ICs

```
ic1 = dde.IC(geom, ..., ..., component=0)
ic2 = dde.IC(geom, ..., ..., component=1)
ic3 = dde.IC(geom, ..., ..., component=2)
```

IV. Integro-differential equations: Volterra IDE

$$\frac{dy}{dx} + y(x) = \int_0^x e^{t-x} y(t) dt, \quad y(0) = 1$$

- kernel

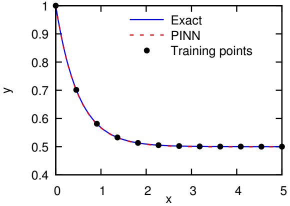
```python
def kernel(x, s):
    return np.exp(s - x)
```

- IDE

```python
def ide(x, y, int_mat):
    rhs = tf.matmul(int_mat, y)  # \int_0^x exp(t-x) y(t) dt
    lhs1 = tf.gradients(y, x)[0]  # dy/dx
    return (lhs1 + y)[: tf.size(rhs)] - rhs
```

- “data”: geometry + IDE + BC + “training” points

```python
data = dde.data.IDE(..., ide, ..., quad_deg=20, kernel=kernel,
    ...)
```



V. Inverse problems

A diffusion-reaction system on $x \in [0,1], t \in [0,10]$:

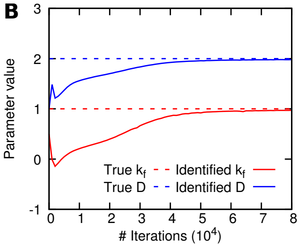
$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2, \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

- Define D and k_f as trainable variables

```
kf = dde.Variable(0.05)
D = dde.Variable(1.0)
```

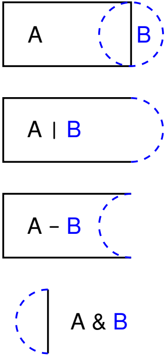
- Define C_A measurements as Dirichlet BC (the same for C_B)

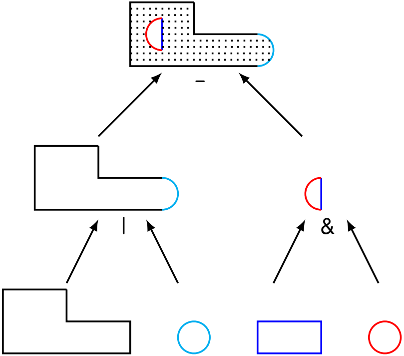
```
observe_x = ...  # All the locations of measurements
observe_Ca = ...  # The corresponding measurements of Ca
observe = dde.PointSetBC(observe_x, observe_Ca, component=0)
```



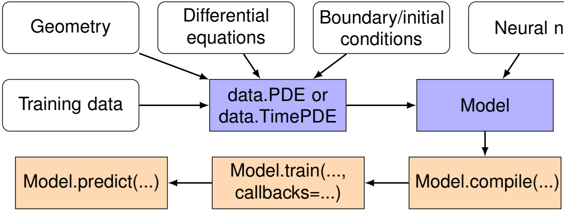
Complex geometry: Constructive solid geometry

- Primitive geometries
  - `interval`, `triangle`, `rectangle`, `polygon`, `disk`, `cuboid`, `sphere`
- boolean operations:
  - Union $A|B$
  - difference $A - B$
  - intersection $A\&B$





DeepXDE



- Short and comprehensive code
- Three backends: TensorFlow 1.x, TensorFlow 2.x, and PyTorch
- No TensorFlow/PyTorch knowledge
- Lots of useful features
- Well-structured and highly configurable

Other functions

- Multi-fidelity NN (**Lu** et al., *PNAS*, 2020)
- DeepONet: Operator learning (**Lu** et al., *Nature Mach Intell*, 2021)

Opening

Lu Group
University of Pennsylvania

- Group website: `https://lu.seas.upenn.edu`
- Chemical and Biomolecular Engineering (CBE)
- Applied Mathematics and Computational Science (AMCS)
- Email: lulu1@seas.upenn.edu