AI in the Sciences and Engineering HS 2025: Lecture 5

Siddhartha Mishra

Computational and Applied Mathematics Laboratory (CamLab)
Seminar for Applied Mathematics (SAM), D-MATH (and),
ETH AI Center (and) Swiss National AI Institute (SNAI) ,
ETH Zürich, Switzerland.

What we have learnt so far ?

- AIM: Learn/Solve PDEs using Deep Neural Networks
- Use PINNs for that purpose.

PINNs for the PDE $\mathcal{D}(u) = f$

- For Parameters $\theta \in \Theta$, $u_\theta : \mathbb{D} \mapsto \mathbb{R}^m$ is a DNN, with $u_\theta \in X^*$
- Aim: Find $\theta \in \Theta$ such that $u_\theta \approx u$ (in suitable sense).
- Compute PDE Residual by Automatic Differentiation:

$$\mathcal{R} := \mathcal{R}_\theta(y) = \mathcal{D}(u_\theta(y)) - f(y), \ y \in \mathbb{D} \quad \mathcal{R}_\theta \in Y^*, \quad \forall \theta \in \Theta$$

- PINNs are minimizers of $\|\mathcal{R}_\theta\|_Y^p \sim \int_{\mathbb{D}} |\mathcal{R}_\theta(y)|^p \, dy$
- Replace Integral by Quadrature !
- Let $\mathcal{S} = \{y_i\}_{1 \le i \le N}$ be quadrature points in $\mathbb{D}$, with weights w_i
- PINN for approximating PDE is defined as $u^* = u_{\theta^*}$ such that

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{i=1}^{N} w_i |\mathcal{R}_\theta(y_i)|^p$$

- Minimize Very high-d Non-Convex loss with ADAM,L-BFGS

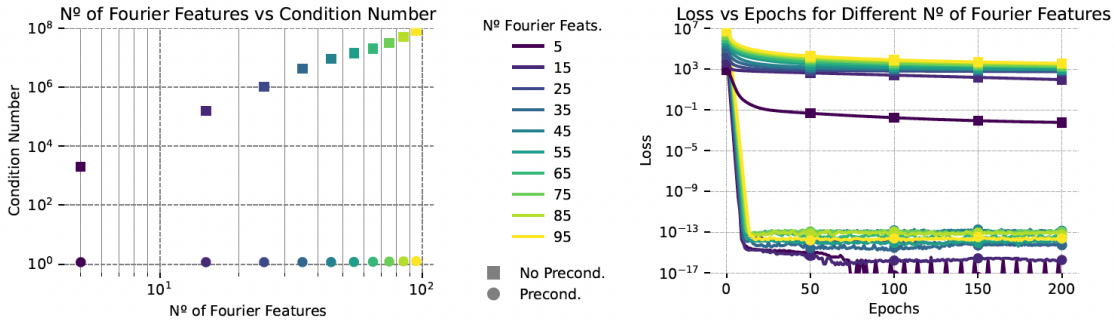
Summary (so far)

- PINNs are observed to empirically very successful for
  - High-dimensional PDEs.
  - Parametric PDEs.
  - Classes of Inverse Problems.
- Theoretical Results For generic PDE: $\mathcal{D}(\mathrm{u}) = f$
- Rigorous Error estimate for PINNs:

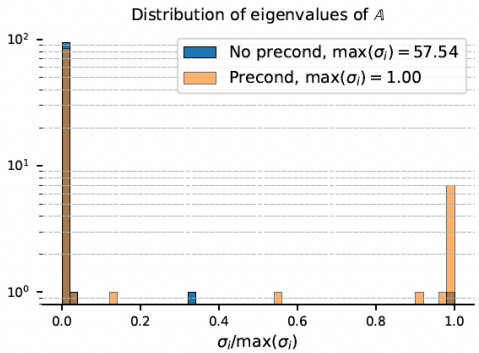
$$\|\mathrm{u} - \mathrm{u}_\theta\| \sim C_{\mathrm{pde}}(\mathrm{u}, \mathrm{u}_\theta)\left[\mathcal{E}_T(\theta) + C_{\mathrm{quad}}(\mathrm{u}_\theta)N^{-\alpha}\right]$$

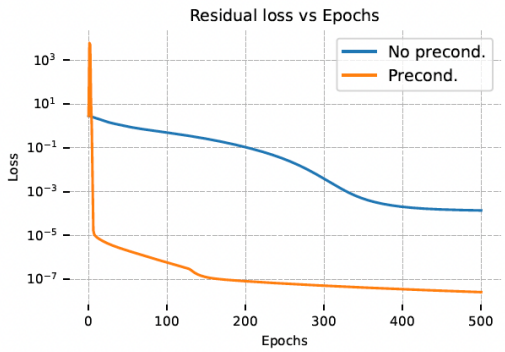
- Caveat I: PINNs may not work for problems with Strong Gradients.
- We have that $\min_\theta \mathcal{E}_T(\theta) \leq \epsilon$
- But can we train to reach close to the global minimum ?
- Caveat II: Training is a Ill-conditioned
- Number of Iterations $\sim \kappa(\mathcal{D}^*\mathcal{D})$

1-D Possion: $-u'' = -k^2 \sin(kx)$

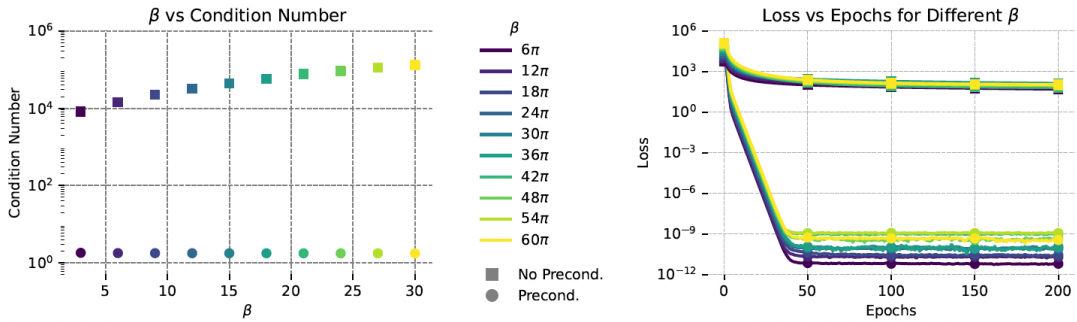


1-D Possion: $-u'' = -k^2 \sin(kx)$

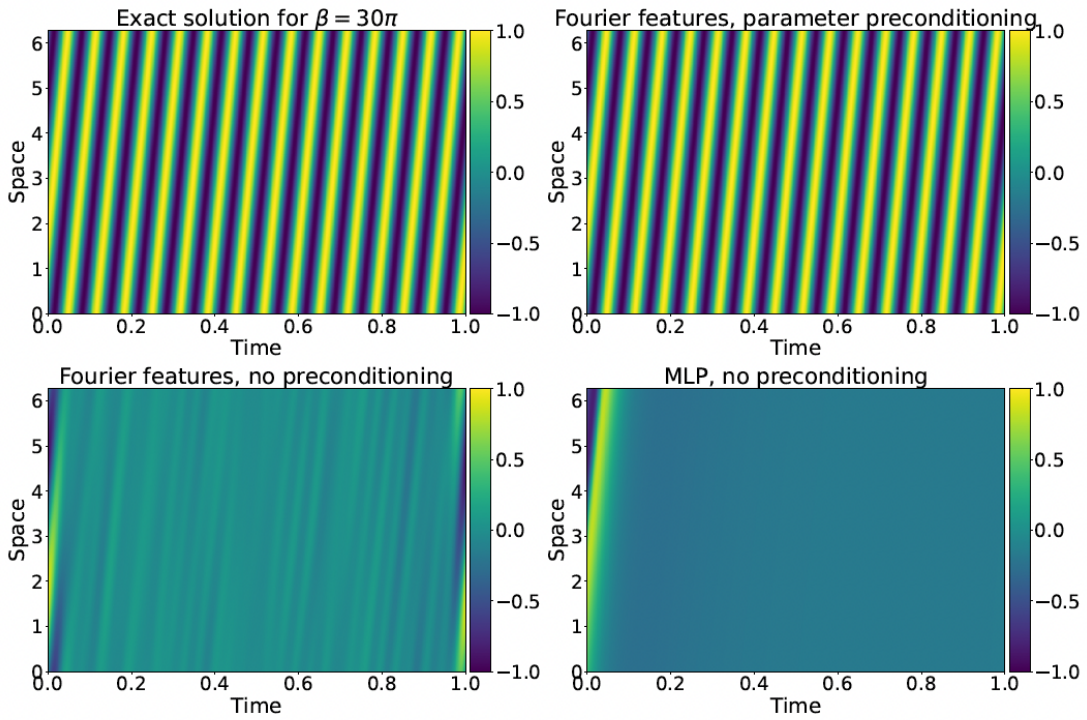




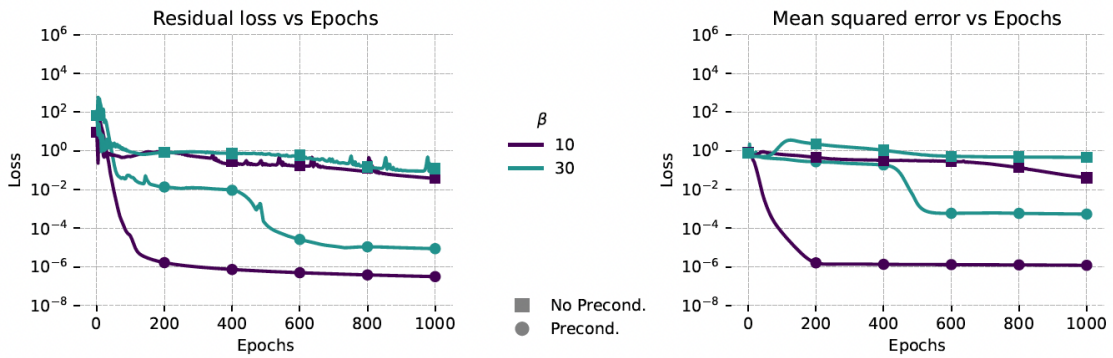
1-D Advection: $u_t + \beta u_x = 0$



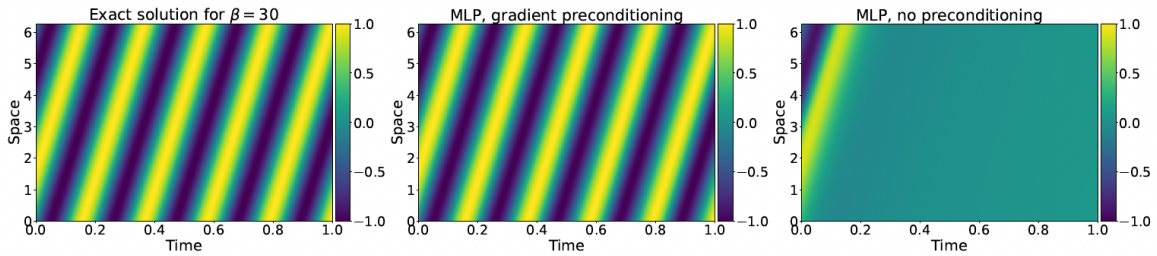
1-D Advection: $u_t + \beta u_x = 0$



1-D Advection: $u_t + \beta u_x = 0$



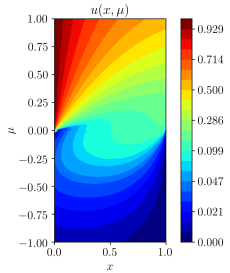
1-D Advection: $u_t + \beta u_x = 0$



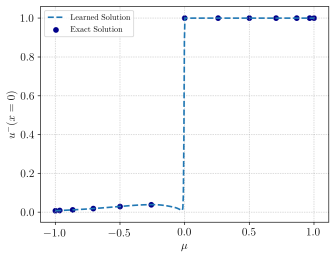
Preconditioning Techniques

- Adjusting balance between Data vs. Physics
- Hard Boundary Conditions (Lagaris et. al)
- Casual Learning (Wang, et. al)
- Second-order Optimizers (Zeinhofer, et. al)
- Multi-stage Neural Networks (Lai et. al)
- Domain Decomposition methods (Moseley et. al)

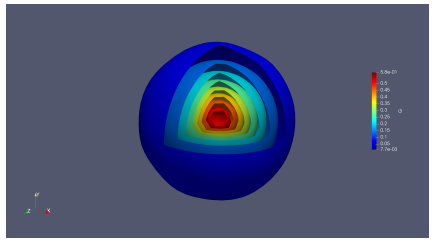
Radiative Transfer



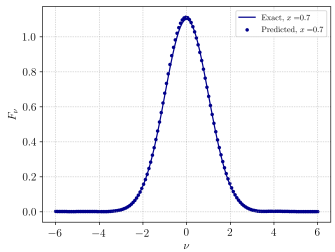
2-D, Intensity



2-D, Boundary



6-D, Inc. Radiation

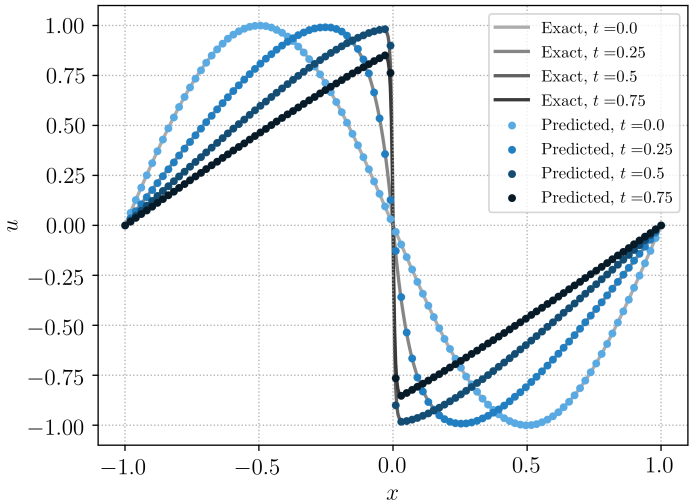


6-D, Radial flux

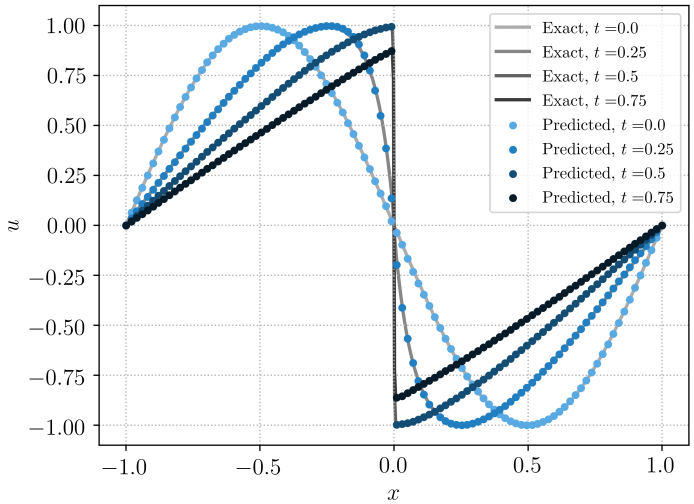
Dimension	Network Size	Error	Training Time
2	24×8	0.3%	57 min
6	20×8	2.1%	66 min

Results for 1-D Burgers'

- Sobol points, $N_{int} = 8192$, $N_{tb} = N_{sb} = 256$, Depth 8, Width 20.



$\nu = 10^{-2}$

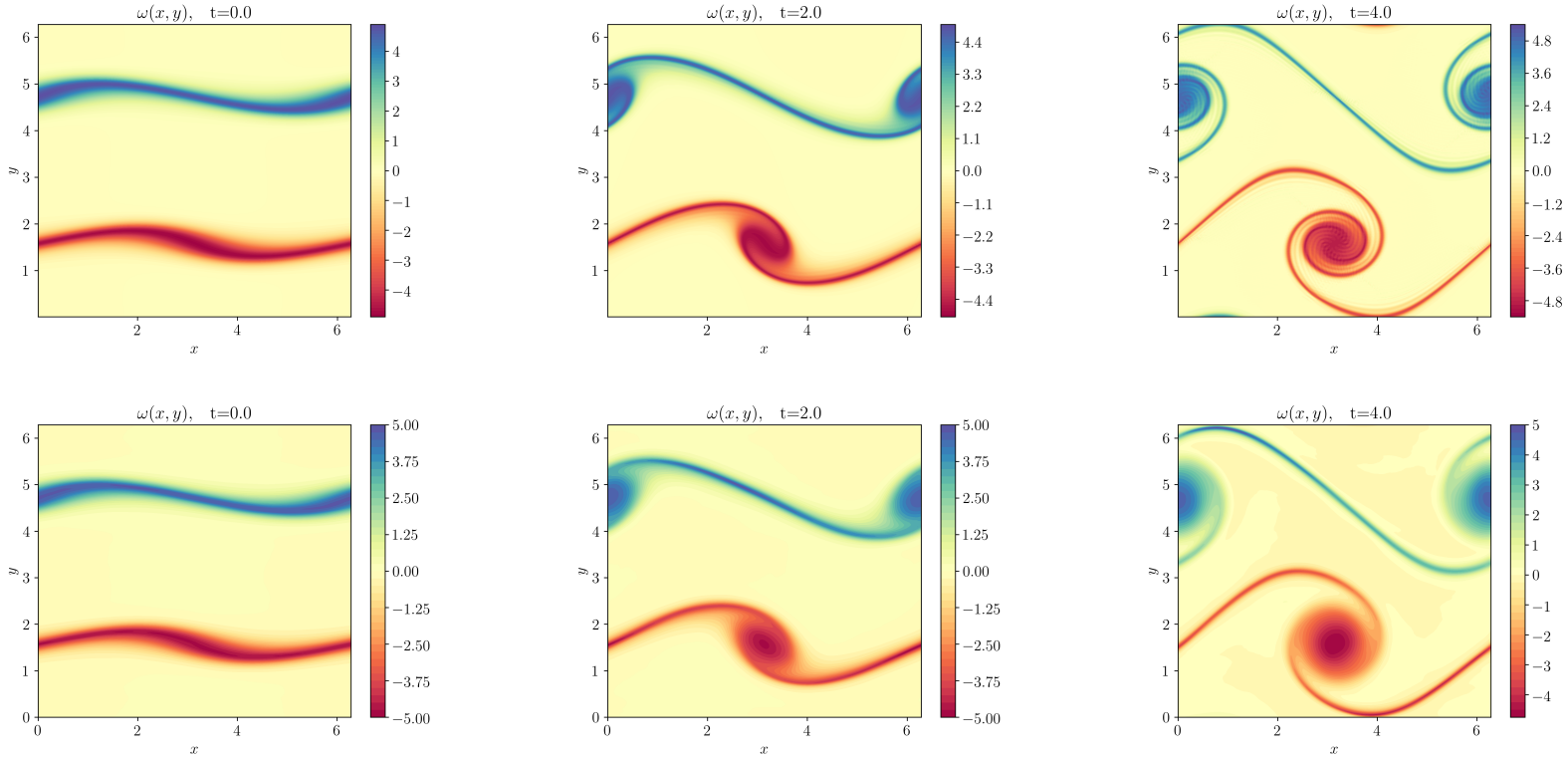


$\nu = 10^{-3}$

Viscosity	Training Error	Total error
10^{-2}	0.0005	1.0%
10^{-3}	0.0008	1.2%

- Finite Difference (0.1 secs) vs. PINNs (5 min)

Results for 2-D Navier-Stokes



- Spectral Method (1 secs) vs. PINNs (30-60 min)

Summary of PINNs.

- PINNs are alternatives to PDE Solvers.
- Work well for "hard" problems with "easy" solutions.
- Don't work yet for complex problems.
- Whats the alternative ?

Summary of PINNs.

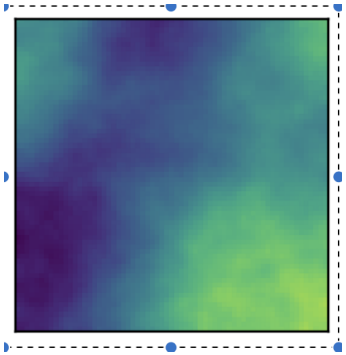
- PINNs are alternatives to PDE Solvers.
- Work well for "hard" problems with "easy" solutions.
- Don't work yet for complex problems.
- Whats the alternative ?
- Use Data:
- The next several lectures: Use of Supervised Deep Learning for PDEs

What does solving a PDE mean ?

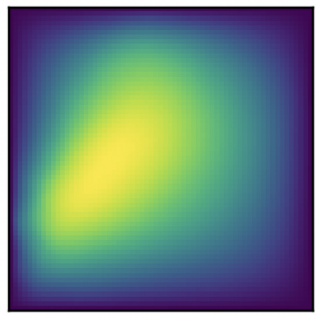
- Example 1: Consider Darcy PDEs:

$$-\mathrm{div}(a\nabla u) = f,$$

- Quantities of interest are:
  - u is temperature or pressure.
  - a is conductance or permeability.
  - f is the source.



a



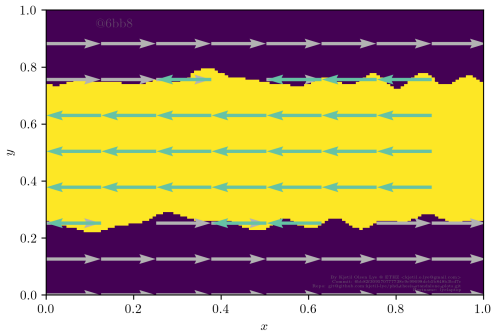
u

- Find the solution Operator $\mathcal{G} : a \mapsto \mathcal{G}a = u$.

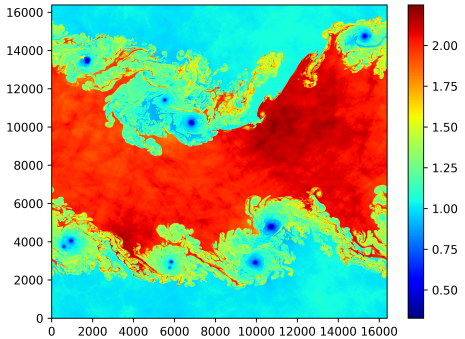
What does solving a PDE mean ?

- Example 2: Consider the Compressible Euler equations:

$$\begin{aligned}
\rho_t + \operatorname{div}(\rho \mathsf{v}) &= 0, \\
(\rho \mathsf{v})_t + \operatorname{div}\left(\rho \mathsf{v} \otimes \mathsf{v} + p\mathsf{I}\right) &= 0, \\
E_t + \operatorname{div}\left((E+p)\mathsf{v}\right) &= 0., \\
u(x,0) = (\rho, \mathsf{v}, E)(x,0) &= a(x).
\end{aligned}$$



Initial Condition



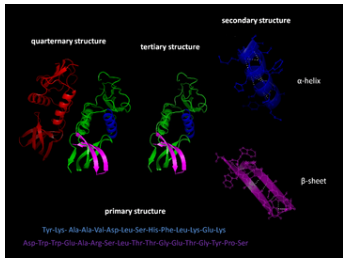
Solution at time T

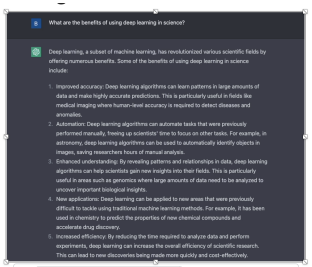
- Find the solution Operator $\mathcal{G} : a \mapsto \mathcal{G}a = u(T)$.

Operator Learning

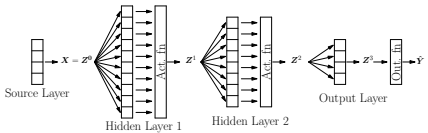
- Operator: $\mathcal{G} : \mathcal{X} \mapsto \mathcal{Y}$, $\dim(\mathcal{X}, \mathcal{Y}) = \infty$.
- Learn PDE Solution Operators from Data
- Underlying Data Distribution $\mu \in \mathrm{Prob}(\mathcal{X})$
- Draw N i.i.d samples $(a_i, \mathcal{G}(a_i))$ with $a_i \sim \mu$.
- Operator Learning Task: Find approximation to $\mathcal{G}_{\#}\mu$

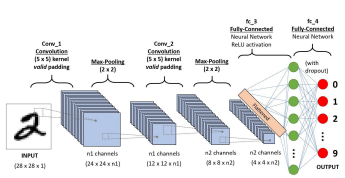
The age of AI

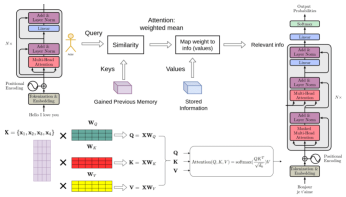




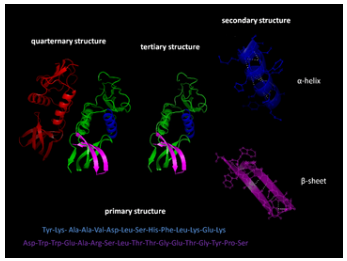
- Exponentially more Compute aka GPUs :-)
- Huge Data
- Deep Neural Networks

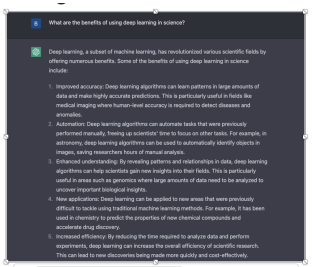




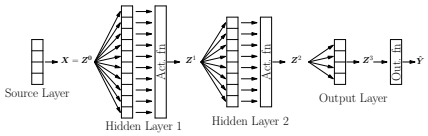


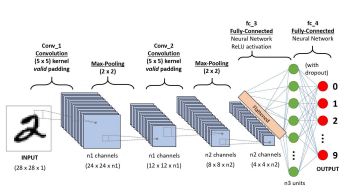
The age of AI

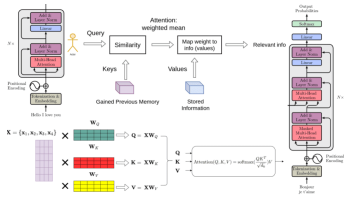




- Exponentially more Compute aka GPUs :-)
- Huge Data
- Deep Neural Networks



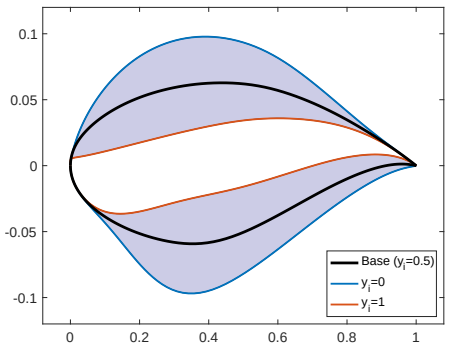


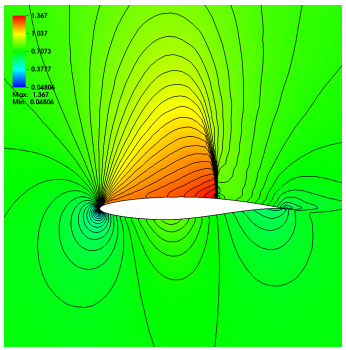


- Neural Networks: $\mathbb{R}^N \mapsto \mathbb{R}^M$
- Caveat: Operator Learning: Inputs+Outputs are Functions !!!

Solution I: Use Parametric PDEs instead :-)

- X, Y are Banach spaces and $\mu \in \mathrm{Prob}(X)$
- Abstract PDE: $\mathcal{D}_a(u) = f$
- Solution Operator: $\mathcal{G} : X \mapsto Y$ with $\mathcal{G}(a, f) = u$
- Simplified Setting: $\dim(\mathrm{Supp}(\mu)) = d_y < \infty$
- Corresponds to Parametrized PDEs with finite parameters.
- Find Soln $u(t, x, y)$ or Observable $\mathcal{L}(y)$ for $y \in Y \subset \mathbb{R}^{d_y}$.



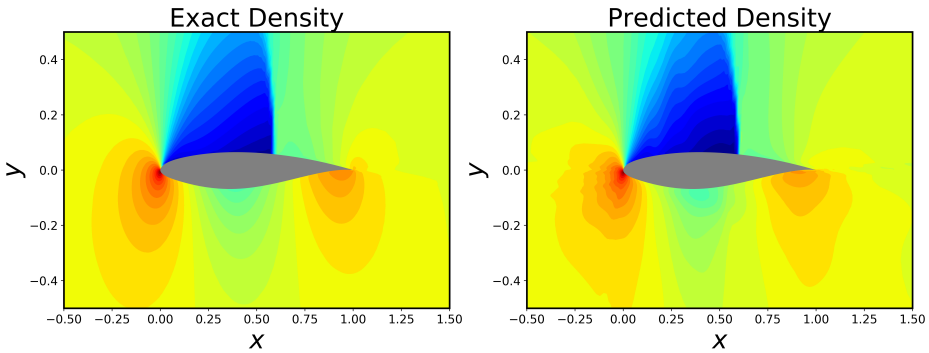


- Approximate Fields or observables with deep neural networks

Works very well

- Given Hicks-Henne parameter: Predict Drag, Lift, Flow
- DNN with $10^3 - 10^4$ parameters and 128 training samples :

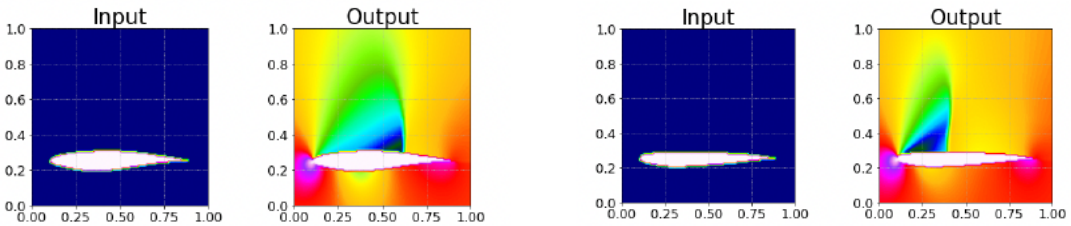
	Run time (1 sample)	Training	Evaluation	Error
Lift	2400 s	700 s	10^{-5} s	0.78%
Drag	2400 s	840 s	10^{-5} s	1.87%
Field	2400 s	1 hr	0.2 s	1.9%



But: A Big Challenge with Parametrization

- Difficult to come up with a suitable one.
- Not unique.
- $f : x \in [0,1] \mapsto x^2$ same as

  $F : (y_1, y_2) \in [0,1]^2 \mapsto \frac{1}{2}\left[\left(\frac{y_1 - b_1}{a_1}\right)^2 + \left(\frac{y_2 - b_2}{a_2}\right)^2\right]$
- Does not Generalize !!



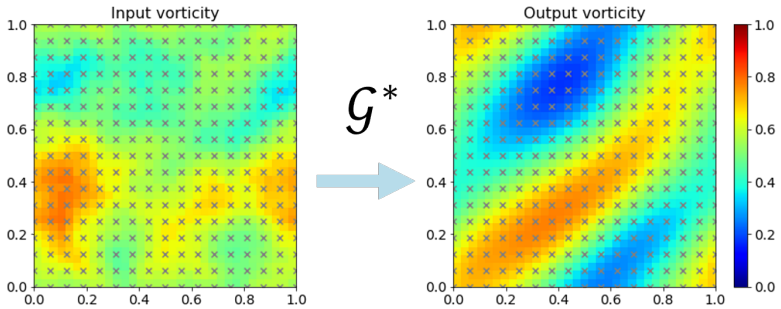
$d = 20$ $d = 30$

- Need to go back to Genuine Operator Learning

Back to Operator Learning

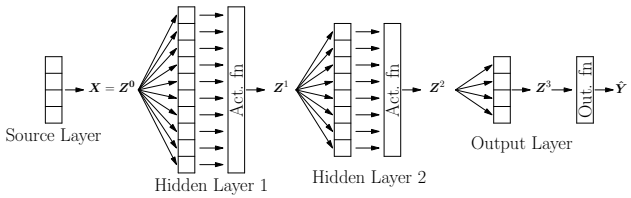
- Operator: $\mathcal{G} : \mathcal{X} \mapsto \mathcal{Y}$, $\dim(\mathcal{X}, \mathcal{Y}) = \infty$.
- Learn PDE Solution Operators from Data
- Underlying Data Distribution $\mu \in \mathrm{Prob}(\mathcal{X})$
- Draw N i.i.d samples $(a_i, \mathcal{G}(a_i))$ with $a_i \sim \mu$.
- Operator Learning Task: Find approximation to $\mathcal{G}_{\#}\mu$

Solution I: Discretize, then Learn !!



- Discretization $\mapsto$ MLP $\mapsto$ Interpolation
- Solution operator $\mathcal{G} : \mathcal{X} \mapsto \mathcal{X}$
- Discretization: $\mathcal{E} : \mathcal{X} \mapsto \mathcal{X}^\Delta \sim \mathbb{R}^N$
- MLP: $\mathcal{L}^* : \mathbb{R}^N \mapsto \mathbb{R}^N$
- Interpolation: $\mathcal{R} : \mathbb{R}^N \mapsto \mathcal{X}$
- Operator Learning: $\mathcal{G}^* = \mathcal{R} \circ \mathcal{L}^* \circ \mathcal{E}$

Deep Neural networks Recalled



- $\mathcal{L}^*(z) = \sigma_o \odot C_K \odot \sigma \odot C_{K-1} \ldots\ldots\ldots \odot \sigma \odot C_2 \odot \sigma \odot C_1(z).$
- At the k-th Hidden layer: $z^{k+1} := \sigma(C_k z^k) = \sigma(W_k z^k + B_k)$
- Trainable Parameters: $\theta = \{W_k, B_k\} \in \Theta$,
- σ: scalar Activation function: ReLU, Tanh
- Random Training set: $\mathcal{S} = \{z_i\}_{i=1}^N \in Z \subset \mathbb{R}^N$, with i.i.d z_i
- Use GD: $\theta_{k+1} = \theta_k - \eta_k \nabla_\theta J(\theta_k)$, to find

$$\theta^* := \arg\min_{\theta \in \Theta} \underbrace{\sum_{i=1}^{N} |\mathcal{L}(z_i) - \mathcal{L}^*_\theta(z_i)|^p}_{J(\theta)},$$

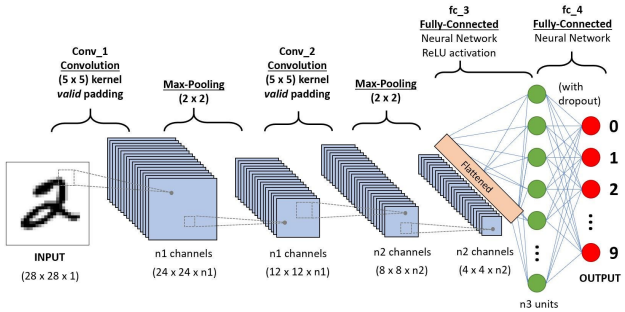
- with $\mathcal{G} = \mathcal{R} \circ \mathcal{L} \circ \mathcal{E}$

Issues

- Large Model Size:
  - For discretization on a 64^2 grid.
  - $N = 4096 \Rightarrow$ Single Hidden Layer of dimension $\mathcal{O}(10^8)$.
  - Too large models even for shallow MLPs.
- Spatial Structure is not incorporated !!
- Not possible to evaluate on a Different Resolution
- Not genuine operator learning.

Solution I (Refined)

- Use CNNs instead of MLP.

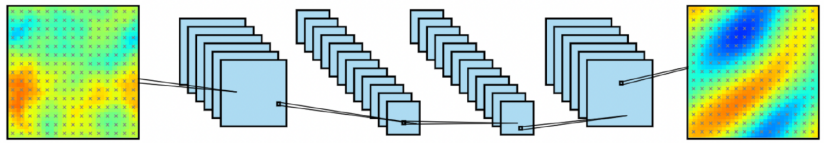


- Key elements are Discrete Convolutions with fixed Kernel:

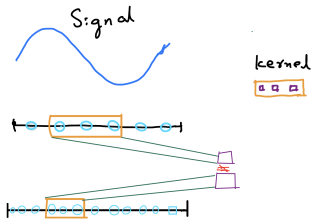
$$K_c[m] = \sum_{i=-s}^{s} k_i c[m-i]$$

- Weight matrices are very Sparse

Operator Learning with CNNs

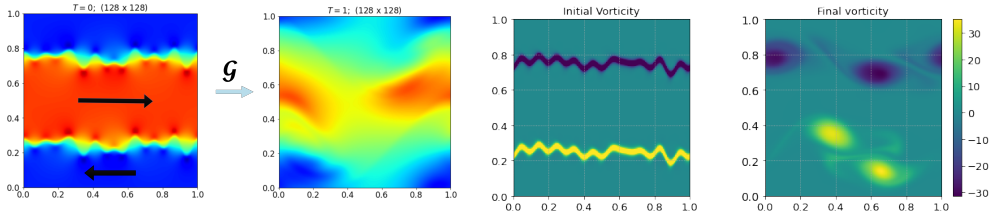


- Tractable Model sizes.
- Respect Spatial structure of underlying Data.
- Possible to Evaluate on any grid resolution



Does this work ?

- Shear flow with Navier-Stokes with $Re >> 1$



- CNN + Interpolation Results:

- Consistent with Zhu,Zabaras, 2019.
- Desiderata for Operator Learning:
- Input + Output are functions.
- Learn underlying Operator, not just a discrete Representation

Solution II: Learn, then Discretize !!

- Use Neural Operators
- Formalized in Kovachki et al, 2021.
- Recall: DNNs are $\mathcal{L}_\theta = \sigma_K \odot \sigma_{K-1} \odot \ldots\ldots \sigma_1$
- Single hidden layer: $\sigma_k(y) = \sigma(A_k y + B_k)$
- Neural Operators generalize DNNs to ∞-dimensions:
- NO: $\mathcal{N}_\theta = \mathcal{N}_L \odot \mathcal{N}_{L-1} \odot \ldots\ldots \mathcal{N}_1$
- Single hidden layer; $\mathcal{N}_\ell : \mathcal{X} \mapsto \mathcal{X}$
- Need to find Function Space versions of
  - Bias Vector
  - Weight Matrix
  - Activation function

Neural Operators (Contd..)

- Replace Bias vector by Bias function $B_\ell(x)$
- Replace Matrix-Vector multiply by Kernel Integral Operators:

$$A_\ell y \to \int_D K_\ell(x,y)v(y)dy$$

- Pointwise activations results in:

$$(\mathcal{N}_\ell v)(x) = \sigma\left(\int_D K_\ell(x,y)v(y)dy + B_\ell(x)\right)$$

- Learning Parameters in B_ℓ, K_ℓ

Discrete Realization

- Caveat: Computational Complexity
- Different Kernels $\Rightarrow$ Low-Rank NOs, Graph NOs, Multipole NOs, ...

Fourier Neural Operators

- FNO proposed in Li et al, 2020.
- Translation invariant Kernel $K(x,y) = K(x-y)$
- Kernel Integral Operator is $\int_D K(x,y)v(y)dy = K * v$
- Key Trick: Perform Convolution in Fourier space
- Fourier Transform: $\mathcal{F} : L^2(D, \mathbb{C}^n) \mapsto l^2(\mathbb{Z}^d, \mathbb{C}^n)$

$$(\mathcal{F}v_j)(k) = \int_D v_j(x)\Psi_k(x)dx, \quad \Psi_k(x) = Ce^{-2\pi i\langle k,x\rangle}$$

- Inverse Fourier Transform: $\mathcal{F}^{-1} : l^2(\mathbb{Z}^d, \mathbb{C}^n) \mapsto L^2(D, \mathbb{C}^n)$

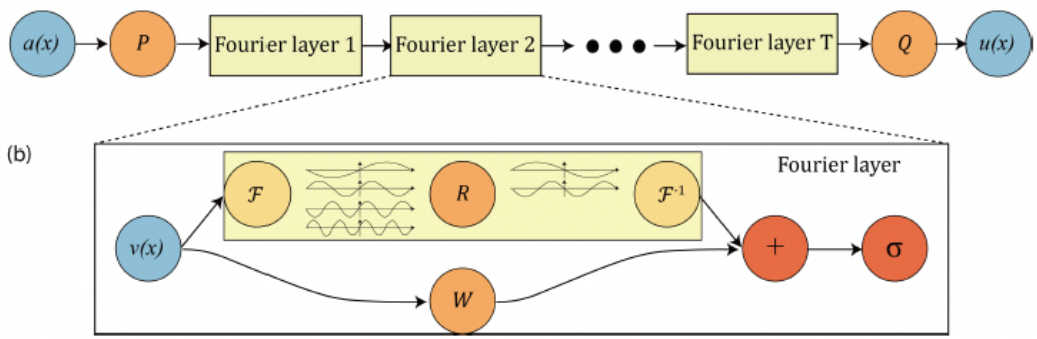
$$(\mathcal{F}^{-1}w_k)(x) = \sum_{k\in\mathbb{Z}^d} w_k\Psi_k(x)$$

FNO Details

- Use Fourier and Inverse Fourier Transform to define the KIO:

$$\int_D K_\ell(x,y)v(y)dy = \mathcal{F}^{-1}(\mathcal{F}(K)\mathcal{F}(v))(x)$$

- Parametrize Kernel in Fourier space.
- With Fixed Number of Fourier Modes
- Fast implementation through FFT



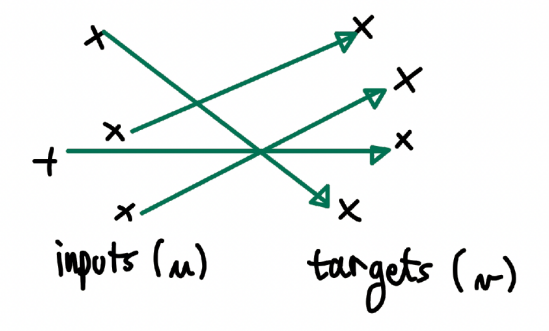
Theory for FNOs

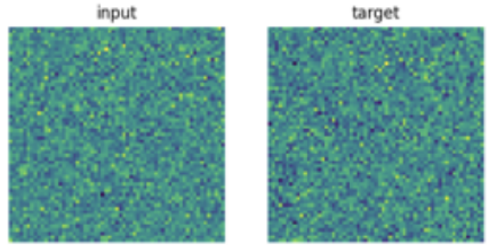
- FNO is very widely used in Practice.
- Theoretical Results of Kovachki, Lanthaler,SM, 2021
- Universal Approximation Thm: For $\mu \in Prob(L^2(D))$ and any measurable $\mathcal{G} : H^r \mapsto H^s$ and $\epsilon > 0$, $\exists \mathcal{N}$ (FNO): $\hat{\mathcal{E}} < \epsilon$
- With Error:

$$\hat{\mathcal{E}}^2 = \int_X \int_U |\mathcal{G}(u)(y)) - \mathcal{N}(u)(y)|^2 \, dy d\mu(u)$$

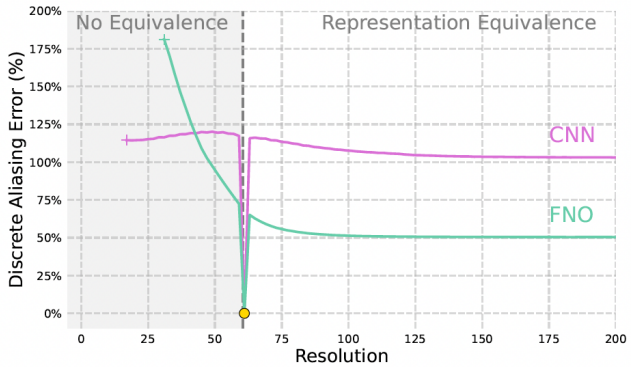
Is there a Catch ?

- A Synthetic Example: Random Assignment
- The underlying Operator:

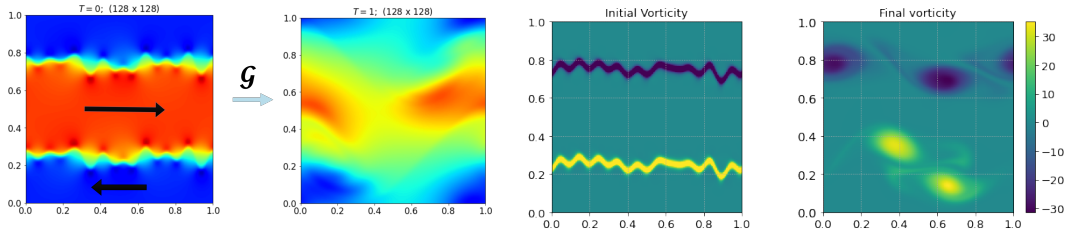




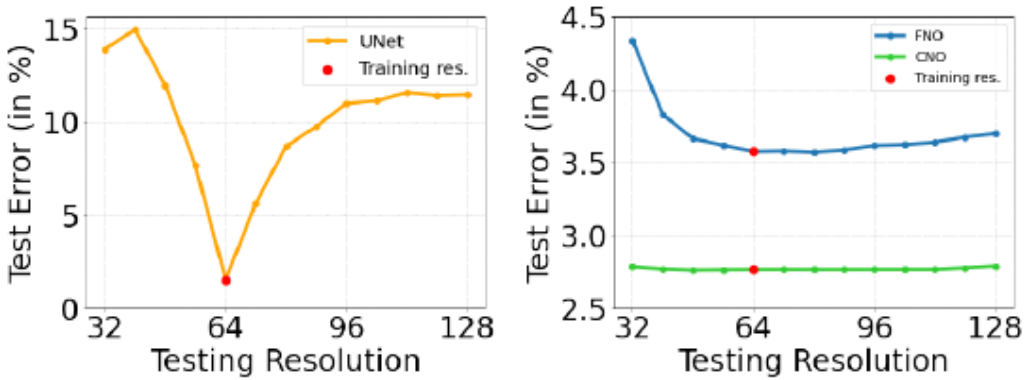
- Errors:



A Practical Example



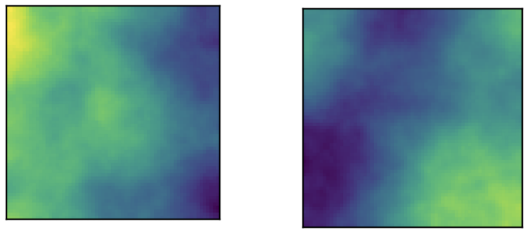
- FNO Results:



- Learn underlying Operator, not just a discrete Representation !!

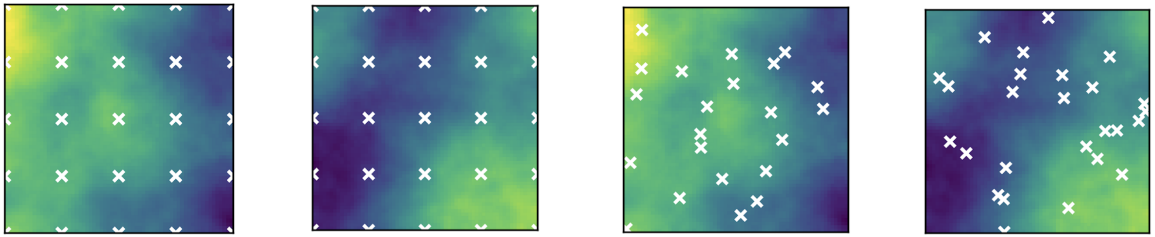
Why is this a challenge ?

- In principle, Operator maps functions to functions.



Input Output

- In practice, both inputs and outputs are Discrete



Input Output Input Output

- Multiple Discrete Representations !!
- Only discrete operations on Digital Computers.
- A proper notion of Continuous-Discrete Equivalence (CDE)