PINNS and Things

Chris Budd[1]

[1]University of Bath

Samba Seminar, March 2025

Some papers to look at

- Grossman et. al. *Can PINNS beat the FE method?*
- Shin et. al. *On the convergence of PINNS for linear elliptic and parabolic PDEs*
- E and Yu *The Deep Ritz Method*
- Appella, B, et. al. *Equidistribution based training of FKS and ReLU Neural Networks*
- Stuart et. al. *Fourier Neural Operator for PDEs*
- Kovachi et. al. *Operator learning: algorithms and analysis*

Motivation: Solving PDEs

Seek to solve PDE problems of the form

$$\mathbf{u}_t = F(\mathbf{x}, u, \nabla u, \nabla^2 u) \quad \text{with BC}$$

eg.

$$-\Delta u = f(x), \quad iu_t + \Delta u + u|u|^2 = 0, \quad u_t = \Delta u + f(x, u).$$

Finite Element Methodology

Express $u(x,t)$ as a Galerkin approximation:

$$u(t,x) \approx U(t,x) = \sum_{i=0}^{N} U_i(t)\, \phi_i(x)$$

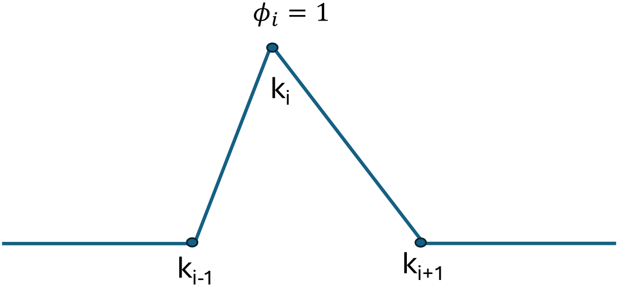
with $\phi_i(x)$ **Not** globally differentiable *locally supported, piece-wise polynomial spline functions*

- Require U to satisfy a *weak form* of the PDE
- Have guaranteed error estimates of the form

$$\|u - U\|_{H^1} < C(u) N^{-\alpha}$$

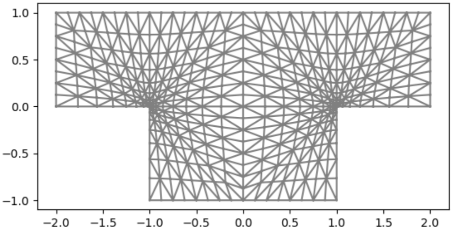
- Can reduce $C(u)$ and increase α using an adaptive approach.
- Awkward in higher dimensions!

Basic linear spline on free knots k_i



Meshes

Traditional PDE computations using Finite Element Methods use a computational mesh τ comprising mesh points and a mesh topology with $\phi_i(x)$ defined over the mesh



Mesh choice

Accuracy of the computation depends crucially on the choice and shape of the mesh

Mesh needs to be

- Fine Enough to capture (evolving) small scales/singular behaviour
- Coarse Enough to allow practical computations
- Able to resolve local geometry eg. re-entrant corners in non-convex domains
- Can inforce structure preserving elements eg. conservation laws.

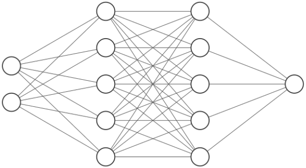
Often hard to find a good mesh!

PINNS

Physics Informed Neural Networks for solving PDEs: advertised as "Mesh free methods".

Use a Deep Neural Net of width W and depth L to give a functional approximation to $u(\mathbf{x})$ with input x.

$$y(\mathbf{x}) = DNN(\mathbf{x})$$



$y(\mathbf{x})$ is constructed via a combination of linear transformations and nonlinear/semi-linear activation functions.

Example: Shallow 1D neural net

$$y(\mathbf{x}) = \sum_{i=0}^{W-1} c_i \sigma(a_i \mathbf{x} + \mathbf{b}_i)$$

Often take

$$\sigma(z) = \mathrm{ReLU}(z) \equiv z_+, \quad \text{or} \quad \sigma(z) = \tanh(z)$$

I: Operation of a 'traditional' PINN

- Assume that $y(\mathbf{x})$ has strong regularity eg. C^2 so not ReLU
- Differentiate $y(\mathbf{x})$ exactly using the chain rule
- Evaluate the PDE residual at collocation points $\mathbf{X}_i$, :chosen to be uniformly spaced, or random
- Train the neural net to minimise a **loss function** L combining the PDE residual and boundary and initial conditions

Eg 1. Solution of regular two-point BVPs by PINNs

Consider the two-point BVP with Dirichlet boundary conditions:

$$-u_{xx} = f(x, u, u_x), \; x \in [0, 1] \quad u(0) = a, \; u(1) = b.$$

Define output of the PINN by $y(x)$ and residual $r(x) := y_{xx} + f(x, y, y_x)$.
The PINN is trained by minimising the loss function

$$L = \frac{1}{N_r} \sum_{i}^{N_r} |r(X_i^r)|^2 + \frac{1}{2} \left(|y(0) - a|^2 + |y(1) - b|^2 \right),$$

where $\{X_i^r\}_i^{N_r}$ are the collocation points placed in $(0, 1)$.

Numerical results for: $-u'' = \pi^2 sin(\pi x)$.

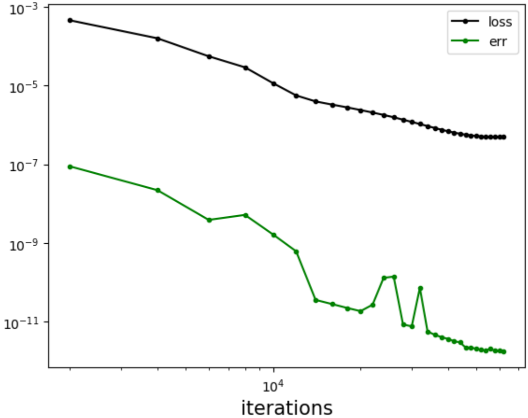


Figure: Residual based Loss and L^2 error of the PINN solution for $N_r = 100$

Eg 2. Singular Reaction-Diffusion Equation

Solve $-\varepsilon^2 u_{xx} + u = 1 - x$ on $[0,1]$ $\quad u(0) = u(1) = 0$

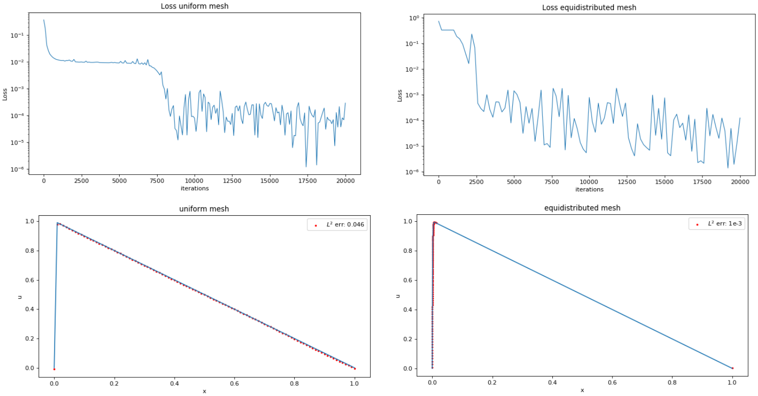


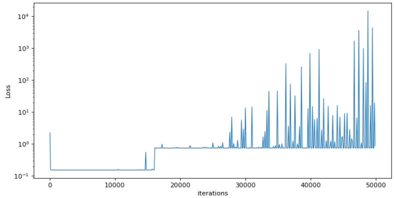
Figure: PINN (tanh) trained for 20000 epochs, $N_r = 101$, Adam optimizer with $lr = 1e-3$. (left) Uniform collocation points (right) Adapted collocation points **much faster training**

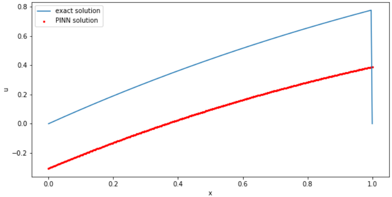
Eg 3. Bad news: Convection-dominated equation

PINNs fail to train when the solution of the BVP exhibits singular and convective behaviour [Krishnapriyan, Aditi et. al., (2021)]:

$$-\varepsilon u_{xx} + \left(1 - \frac{\varepsilon}{2}\right) u_x + \frac{1}{4}\left(1 - \frac{1}{4}\varepsilon\right) u = e^{-x/4} \text{ on } [0,1] \quad u(0) = u(1) = 0$$

$$u(x) = exp^{\frac{-x}{4}}\left(x - \frac{exp^{-\frac{1-x}{\varepsilon}} - exp^{-\frac{1}{\varepsilon}}}{1 - exp^{-\frac{1}{\varepsilon}}}\right)$$





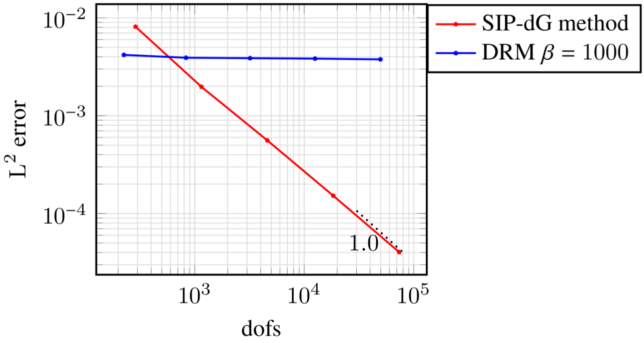
II Operation of a 'variational' PINN such as a Deep Ritz Method (DRM) [Weinan E et. al.]

- Assume that $y(\mathbf{x})$ has weaker regularity eg. H^1 So ReLU is OK
- Differentiate $y(\mathbf{x})$ exactly using the chain rule
- Construct an appropriate weak form of the PDE (typically involving an integral)
- Evaluate the weak form by using quadrature at quadrature points $\mathbf{X}_i$ (chosen to be uniformly spaced, or random)
- Train the neural net to minimise a loss function combining the weak form and boundary and initial conditions

Example: Poisson equation in 2D

DRM method works well for small DOF

dG Finite Eelement Method is **much** better for more DOF



This convergence pattern is seen in many other examples

General questions for consideration

1. When do and don't PINNs work, and why?
2. How do these answers depend on (i) the problem (ii) choice of activation fiunction, optiisation, collocation points etc
3. Can we develop a useful convergence theory for a PINN using tools from approximations theory, bifurcation theory, numerical analysis etc.
4. How does a PINN compare to a finite element method?

Comparing a PINN to an (adaptive) finite element method

If $\sigma(z) = ReLU(z)$ then

$$y(\mathbf{x}) = \sum_{i=0}^{W-1} c_i (a_i \mathbf{x} + \mathbf{b}_i)_+$$

In 1-dimension this is a **piece-wise linear function** with N **free knots** at

$$k_i = -b_i / a_i.$$

[deVore] Any 1D ReLU network of width W and depth L is formally equivalent to a piecewise linear free knot spline (FKS) approximationn with $N \ll W^L$ free knots k_i, where

$$y(x) = \sum_{i=0}^{N} w_i \phi_i (x - k_i).$$

Similar results but **MUCH** more complex in higher dimensions.

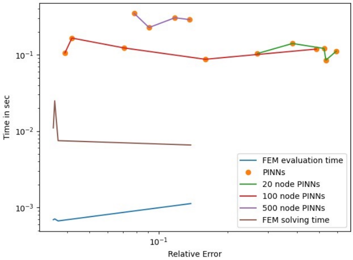
Expressivity of a PINN

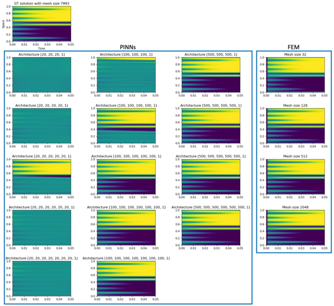
SO .. in principle a ReLU network has the same expressivity as an adaptive Finite Element Method and should deliver the same error estimates if correctly trained.

Compare with a traditional linear spline (used in FE) which is often a **piecewise linear Galerkin approximation to a function with a fixed mesh.** Good convergence, but often much slow than an adaptive FE method and hence a well trained PINN

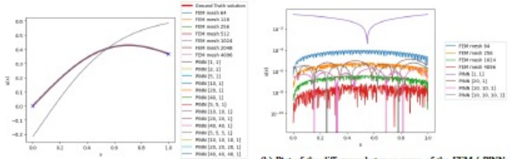
BUT do we ever see this in practice?

(Solution of the Allen-Cahn Equations





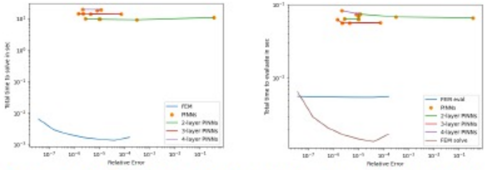
Results by Grossman et. el. 2



(a) Plot of the PDE ground truth solution and the FEM and PINNs solution approximations.

(b) Plot of the difference between some of the FEM / PINNs solution approximations to the ground truth solution on a log scale.

Figure 1: Plot for 1D Poisson equation solution.



(a) Plot of time to solve FEM and train PINN in sec versus ℓ^2 relative error.

(b) Plot of time to interpolate FEM and evaluate PINN in sec versus ℓ^2 relative error.

Figure 2: Plot for 1D Poisson equation of time in sec versus ℓ^2 relative error.

FE: linear

- Limited expressivity, reduced accuracy
- Adaptive only with effort
- Not equivariant
- Need a complex mesh data structure
- Convex with guarantees of uniqueness for many problems (and direct calculation using linear algebra)
- Work on saddle-point problems (eg. most problems)
- Good (a-priori and a-posteriori) guaranteed error bounds :

  Cea's Lemma: Bounds solution error by interpolation error on the FE space

DRM:nonlinear

- Very expressive (potential high accuracy for a small number of degrees of freedom)
- Self adaptive
- Equivariant
- Don't need a complex mesh data structure
- Don't work on saddle-point problems (eg. most problems)

Start of a convergence theory for PINNS

[Shin, Darbon and Kaniardarkis, 2020], [Jiao, Lai, Lo, Wang, Yang]

- PINN error is a combination of approximation error and training/optimization error
- Show that a PINN (depth L width W) can be constructed with low approximation error which reduces as the complexity of the PINN increases.
- Hope that the optimization error can be reduced to acceptable levels.
- Try things out on simple problems

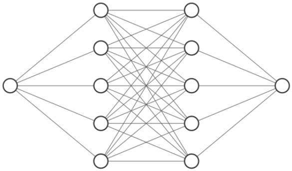
BUT

- **Non convex (no guarantee of uniqueness or convergence of training)**
- PINNs are nonlinear function approximators. No equivalent of Cea's lemma giving a bound on the solution error.
- Solutions can sometimes have no connection to reality!

ReLU Neural Networks and approximation in 1D

Convergence of a PINN/DRM relies on understanding **both** approximation and training

A feed-forward Deep Neural Network (DNN) can be 'in principle' trained to approximate a target function $u(x)$



$$y_j = \Sigma_{i=0}^{W-1} c_{i,j} \, \sigma(a_{i,j} y_j + b_{i,j}) \; j = 1, \cdots, L \quad y_0 \equiv x, \quad y(x) = y_L$$

Expressivity

There are a lot of results on the **theoretical expressivity** of a DNN [Grohs and Kutyniok, Mathematical aspects of deep learning, (2022)].

Universal approximation theorem, [Hornick et. al., 1989] *If* $u(\mathbf{x})$ *is a* continuous function *with* $\mathbf{x} \in K \subset R^d$ *(compact), and* σ *a continous activation function. Then for any* $\epsilon > 0$ *there exists a (shallow) neural network* $y(\mathbf{x}, \theta)$ *such that*

$$\|u(\mathbf{x}) - y(\mathbf{x})\|_\infty < \epsilon.$$

Theorem [Yarotsky (2017)]
Let

$$F_{n,d} = \{u \in W^{n,\infty}([0,1]^d) : \|u\|_{W^{n,\infty}} \leq 1\}$$

For any d, n *and* $\epsilon \in (0,1)$ *there is a ReLU network architecture that*

1. *Is capable of expressing any function from* $F_{d,n}$ *with error* ϵ
2. *Has depth*

$$\text{depth:} \quad L < c(\log(1/\epsilon) + 1) \quad \text{width:} \quad W < c\epsilon^{-d/n}(\log(1/\epsilon) + 1)$$

for some constant $c(d, n)$*.*

Implies exponential rates of convergence with increasing depth L

But how well do we do in practice?

Direct Learned Univariate Function Approximation

For a learned function $y(x)$ approximate target function $u(x)$ by minimising the 'usual' loss function L over parameters θ

$$\min_{\theta} \; L(\theta) \equiv \sum_{k=1}^{M} |y(X_k) - u(X_k)|^2$$

Use the shallow ReLU network

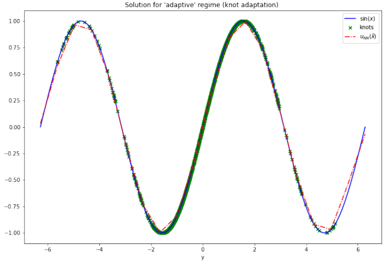
$$y(x) = \sum_{j=0}^{W-1} c_j(a_j x + b_j)_+, \quad \theta = [\mathbf{a}, \mathbf{b}, \mathbf{c}].$$

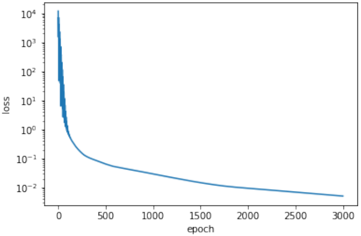
Find the optimal set of coefficients a, b, c

Use an ADAM SGD (over the quadrature points) optimiser

Approximation of: $u(x) = \sin(x)$ using ReLU network

Uniform quadrature points:

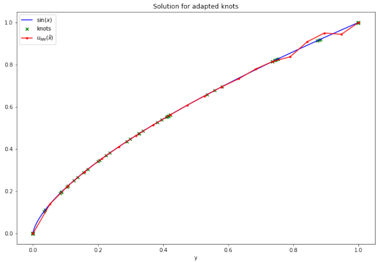


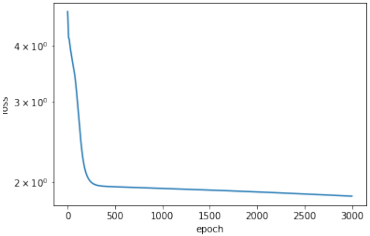


Results poor. Depend crucially on the starting values. Even then poor.

Approximation of: $u(x) = x^{2/3}$

Uniform quadrature points:





Results even worse! Depend crucially on the starting values. Even then very bad!

Problems with:

- the loss function,
- training,
- and conditioning,

of the ReLU NN.

REASON: Trying to train a_i, b_i, c_i together, whereas they play very different role in the approximation.

This leads to a very non-convex and ill conditioned optimisation problem

RESOLUTION:

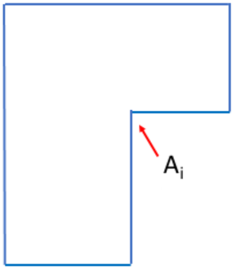
- **First** solve the *nonlinear* problem of finding nearly optimal (knots) a_i, b_i first
- **Then** Solve the *nearly linear but ill-conditioned* problem of finding the (weights) c_i next by **pre-conditioning the system**
- **Iterate** if needed (it's not needed)

This method uses a lot of finite element theory to work and delivers good approximations. But much work needed to make it work for PINNS in general.

Poisson Problem on an L-shaped domain

Problem to solve:

$$\boxed{\begin{aligned} -\Delta u &= f \text{ in } \Omega \\ u &= u_D \text{ on } \Gamma_D \\ \nabla u \cdot \vec{n}_\Omega &= g \text{ on } \Gamma_N. \end{aligned}}$$



Singular solution

- Solution $u(\vec{x})$ has a gradient singularity at the interior corner A_i
- If the interior angle is ω and the distance from the corner is r then

$$u(r,\theta) \sim r^{\alpha} f(\theta), \quad \alpha = \frac{\pi}{\omega}$$

where $f(\theta)$ is a regular function of θ

- Corner problem

$$u(r,\theta) \sim r^{2/3}, \quad r \to 0.$$

Numerical results: **random** quadrature points

Solve $\Delta u(x) = 0$ on $\Omega_L \quad u(r,\theta) = r^{2/3} sin(2\theta/3)$ on $\Gamma = \partial\Omega_L$

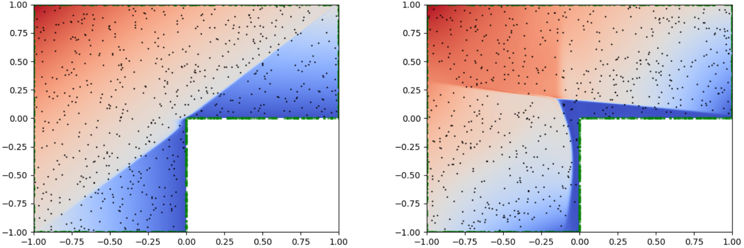


Figure: Left: PINN Right: DRM

Can we improve the accuracy by a better choice of collocation/quadrature points?

Optimal collocation points for the L-shaped domain

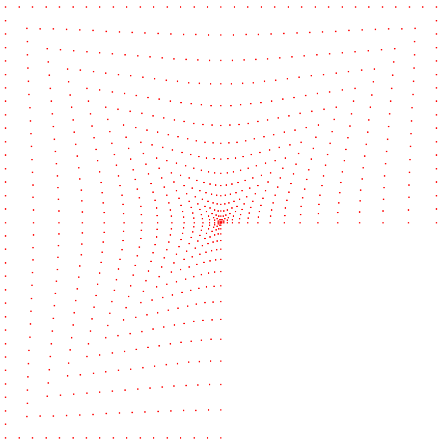
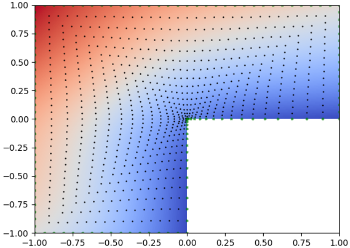


Figure: Optimal points for interpolating $u(r,\theta) \sim r^{2/3}$

Optimal points and PINN/Deep Ritz

Solutions with Optimal quadrature points



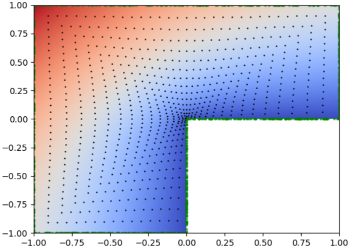


Figure: L^2 error - randomly sampled points: 0.468 | Optimal: 0.0639

Left: PINN, Right: Deep Ritz

Good choice of quadrature points makes a big difference, but still problems with pre-conditioning

Operator based methods such as FNO

Idea: Have an evolutionary PDE defined for $x \in \Omega$

$$u_t = F(x, t, u, u_x, u_x x), \quad u(0, x) \equiv u_0(x) \in H^1(\Omega).$$

If $u_1(x) \equiv u(1, x)$ this *can* induce a *map* $N : H^1(\Omega) \to H^1(\Omega)$

$$N : u_0 \equiv u_1.$$

- If F is linear then so is N. Otherwise nonlinear
- If F is Lipschitz then N is continuous. Otherwise properites of N are unclear and it may not even exist!
- If u satisfies a conservation law then so does N.

Neural operator methods try to learn the map N. Most literature is for the linear, Lipschitz case.

Supervised Learning

- Have a Neural Net with input (a discretised form of) $u_0(x)$ and output $\Psi(u_0)$ (a discretised approximation of) $u_1(x)$ and parametrised by θ.
- Using your favourite PDE solver (eg. pseudo-spectral method, FE etc.) generate lots of input-output pairs $(u_{0,i}(x), u_{1,i})(x)$ for $i = 0 \ldots N$ which you hope span a relevant subset of the (infinite dimensional) function space $H^1(\Omega)$.
- Construct a loss function eg.

$$L = \| u_{1,i} - \Psi(u_{0,i}) \|.$$

- Optimize θ to minimize the loss over the training set using SGD/Adam. Validate in the usual way
- Can then apply the Neural Operator NN to find u_1 for *arbitrary* initial data without having to solve the PDE. eg. Weather forecasting

Notes

- Many different architectures for NN. Many make use of latent finite dimensional structures. Examples include DeepONet and FNO.
- This is supervised learning of the operator as we are generating the pairs in advance. It differs significantly from the semi-supervised training of a PINN.
- Quality of the NN depends significantly on the quality of the training set.
- Various theorems exist on the convergence/accuracy of the Neural operator. They rely on properties of N which depend in a very subtle way on the properties of the underlying PDE. See [Kovachi et. al.]

Example

Simplest example is the linear heat equation

$$u_t = u_{xx}, \quad x \in \Omega.$$

Then have if $\Omega = R$

$$N : u_0 \to u_1 \equiv G * u_0 \equiv \frac{1}{\sqrt{2\pi t}} \int_{-\infty}^{\infty} e^{-(x-y)^2/t} u_0(y)\, dy.$$

Also if u satisfies *periodic boundary conditions* $x \in [0, 2\pi]$

$$N : u_0(x) \equiv \sum_n a_n\, e^{inx} \to u_1(x) \equiv \sum_n e^{-n^2}\, a_n\, e^{inx}.$$

So the linear map N is defined in terms of integral operators and simple spectral operations.

The FNO

The FNO architecture is based on the process of solving the linear heat equation:

$$\Psi(u,\theta)_{FNO} \equiv Q \circ \mathcal{L}_L \circ \ldots \circ \mathcal{L}_2 \circ \mathcal{L}_1 \circ R(u).$$

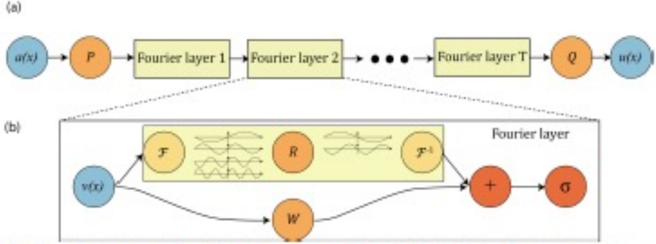
$$\mathcal{L}_n(v)(x,\theta) = \sigma(W_n v(x) + b_n + K(v))$$

Here W is a pointwise linear local map. $K(v)$ is a global integral operator, kernal $G_n(\theta)$. Evaluate Kv using an FFT via

$$FFT(Kv) = FFT(G_n)\, FFT(v).$$

FFT restricted to M modes. Nonlinearity and higher order modes introduced via the activation function σ.

Figure from FNO paper



(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers:** Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

Figure 2: **top:** The architecture of the neural operators; **bottom:** Fourier layer.

Examples:

2D Allen-Cahn equation modeling phase separation:

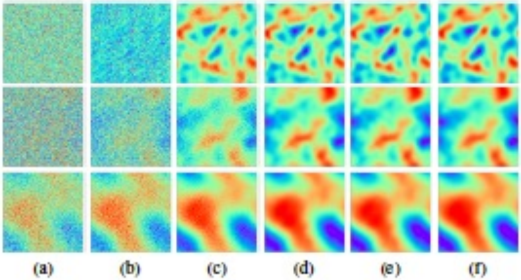
$$\begin{aligned} u_t &= u - u^3 + \epsilon^2 \Delta u, \quad && \mathbf{x} \in [0,1]^2, t \in (0, T) \\ u(0,\mathbf{x}) &= u_0(\mathbf{x}), && \mathbf{x} \in [0,1]^2 \end{aligned} \tag{1}$$

Navier-Stokes eqns (vorticity formulation)

$$\begin{aligned} &\omega_t = -u \cdot \omega_x - v \cdot \omega_y + \nu \Delta \omega + f, \quad \mathbf{x} \in [0,1]^2, t \in (0, T) \\ &\omega = v_x - u_y, \quad w(0,\mathbf{x}) = w_0(\mathbf{x}), \quad \mathbf{x} \in [0,1]^2, \end{aligned} \tag{2}$$

Results 1

Predictions of FNO trained on different datasets for the Allen-Cahn equation ($\epsilon = 0.05$). (a) inputs with noise (b) 1000 data pairs (c) 10000 data pairs (d) 1000 data pairs + 1000 generated data pairs (e) 5000 data pairs + 5000 generated data pairs (f) ground truth [with Chaoyu Liu]



Results 2

Test error of neural operators on PDE datasets with 1000 and 10000 training samples.

PDE Dataset	Training Samples	FNO	UNO	CNO	UNet-FNO
2*Darcy Flow	1000	3.157e-2	3.141e-2	2.295e-2	**1.312e-2**
	10000	1.686e-2	1.770e-2	1.034e-2	**7.142e-3**
2*Navier-Stokes ($\nu = 1e-3$)	1000	7.940e-3	7.775e-3	1.090e-2	**4.250e-3**
	10000	2.626e-3	1.937e-3	2.016e-3	**1.204e-3**
2*Navier-Stokes ($\nu = 1e-4$)	1000	9.410e-2	9.282e-2	6.701e-2	**4.135e-2**
	10000	5.271e-2	5.617e-2	2.036e-2	**1.570e-2**
2*Allen-Cahn ($\epsilon = 0.05$)	1000	1.376e-2	1.646e-2	2.980e-2	**5.008e-3**
	10000	2.945e-3	2.967e-3	1.321e-2	**2.060e-3**
2*Allen-Cahn ($\epsilon = 0.01$)	1000	1.050e-2	1.511e-2	1.910e-2	**3.347e-3**
	10000	7.098e-3	1.173e-2	9.981e-3	**1.135e-3**
2*Compressible Navier-Stokes	1000	2.740e-1	2.846e-1	5.644e-1	**2.355e-1**
	10000	2.330e-1	2.089e-1	2.911e-1	**1.640e-1**

Areas for improvement and research

- NONE of this applies, for example, to the nonlinear heat equation

$$u_t = u_{xx} + u^2.$$

- Observe poor conservation laws at the moment
- Generating a good training set is crucial and can be slow. How to make it good and fast?
- FNO struggles away from the traning set. Need to broaden its scope and extend the theorems on its convergence

Summary

- PINNS and NOs both show promise as a quick way of solving PDEs but have only really been tested on quite simple problems so far
- PINS not (yet) competitive with FE in like-for-like comparisons
- PINNs need careful meta-parameter tuning to work well
- NOs proving more promising. Now used for weather forecasting!
- Long way to go before we understand PINNS or NOs completey and have a satisfactory convergence theory for them in the general case.
- Lots of great stuff to do!