PHYSICS INFORMED NEURAL NETWORKS

Some Insights and Limitations

MathsInFluid, ENS Lyon

Outline of the Talk

- Disclaimer
- Context : At the interface of two modeling paradigms
- What is a physics informed neural network ?
- Some examples + a minimal working Pytorch code
- Some Theoretical Results :
  Approximation, Quadrature, ***Training errors***
- Some tricks and Techniques to get it to work,

Context

Two modeling paradigms

Physical modeling

- based on deep understanding of system,
- stems from first principles (PDEs),
- **difficult to solve**,
- not made to scale with data,

Statistical modeling

- little prior information,
- knowledge is extracted from data,
- very simple to develop,
- scales very well with data.

Combining both Paradigms

Physics-Informed Machine Learning

- leverage prior information + extract knowledge from data,
- make the best of both worlds ?

Challenges :

- develop fast surrogate models,
- augment physical models,
- handle partial information or observations, etc...

Combining both Paradigms

Physics-Informed Machine Learning

- leverage prior information + extract knowledge from data,
- make the best of both worlds ?

Challenges :

- develop fast surrogate models,
- augment physical models,
- handle partial information or observations, etc...

Physics-Informed Neural Networks

- Started with Lagaris et al. [1998], popularized by Raissi et al. [2019]
- Combines PDE and NN through the loss function,
- Made to solve forward + inverse problems,

As of 2021..

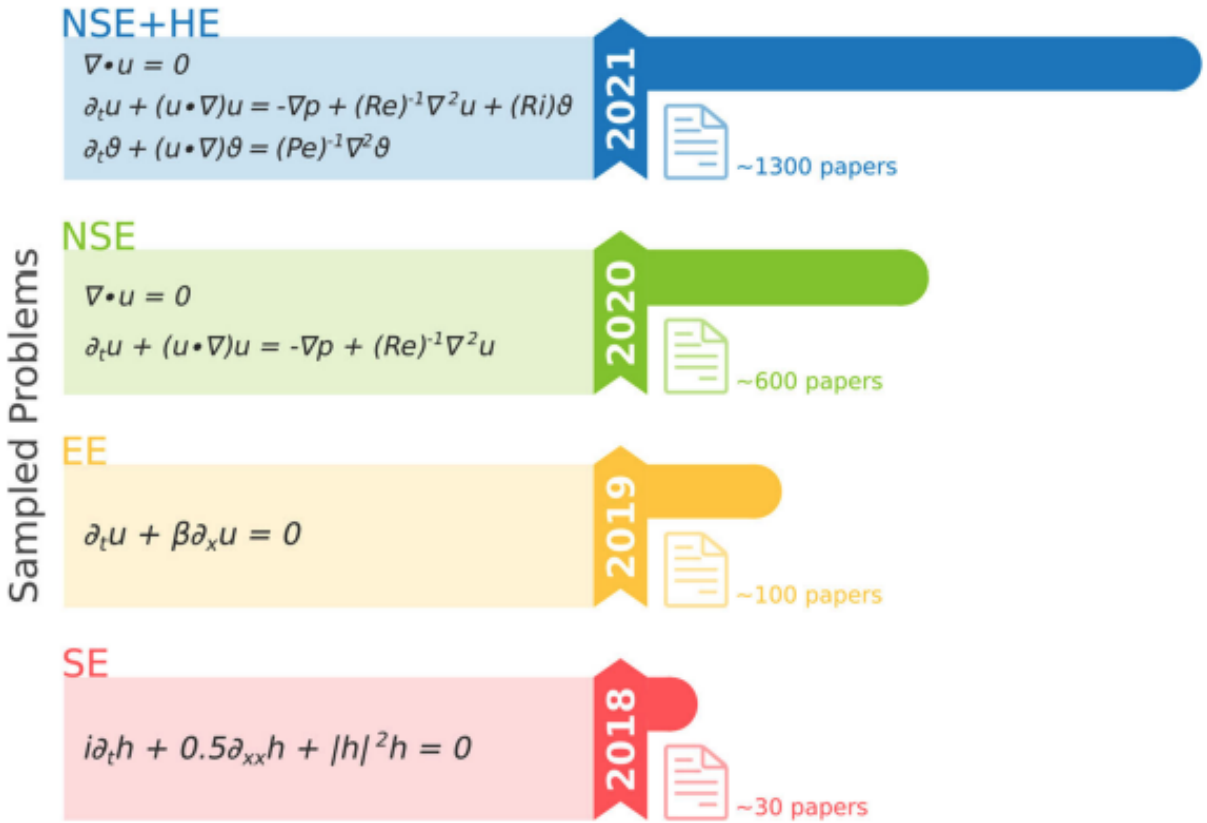


Figure 1 – From Cuomo et al. [2022]

What is a Physics-Informed Neural Network ?

What is a Physics-Informed Neural Network ?

Partial Differential Equation

- $\mathcal{D}(u) = f, \quad x \in \Omega$
- $\mathcal{B}(u) = 0, \quad x \in \partial\Omega$

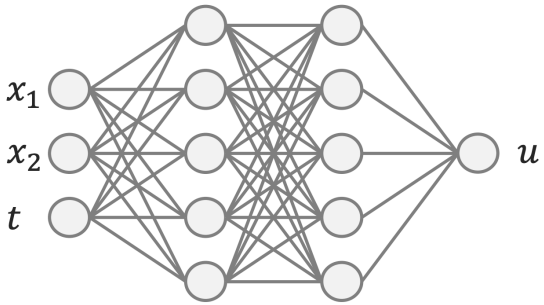
Deep Learning

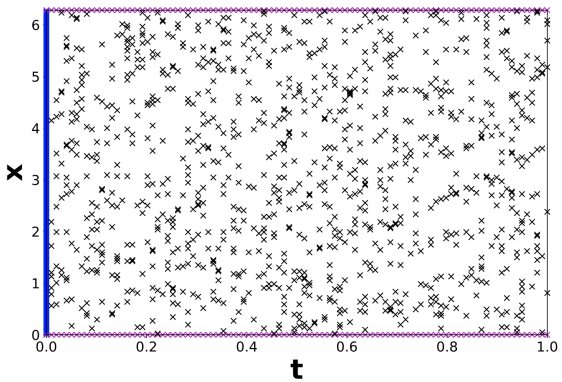
- u_θ is a Neural network,
- loss function $\mathcal{L}(\theta)$
- $\theta_{k+1} = \theta_k - \eta \nabla \mathcal{L}(\theta_k)$

Physics-Informed Neural Networks [Raissi et al., 2019]

- Neural network $\quad u_\theta : \Omega \to \mathbb{R}^d, \quad x \mapsto u_\theta(x)$
- $\mathcal{L}(\theta) = \int_\Omega |\mathcal{D}(u_\theta)(x) - f(x)|^2 dx \; + \; \lambda \cdot \int_{\partial\Omega} |\mathcal{B}(u_\theta)(x)|^2 dx,$
- $\mathcal{D}(u_\theta)$ computed from $\mathrm{Jac}_x(u_\theta), \mathrm{Hess}_x(u_\theta)$ using *autograd*,
- $\mathcal{L}(\theta)$ minimized using SGD and variants,
- If $\mathcal{L}(\theta) = 0$ then u_θ is solution of the PDE.

A minimal Example : Solving the 2d-Heat Equation





Monte-Carlo approximation of $\mathcal{L}(\theta)$:

$$\hat{\mathcal{L}}(\theta) = \sum_i \left| \frac{\partial u_\theta}{\partial t}(x_i, t_i) - D \frac{\partial^2 u_\theta}{\partial x^2}(x, t_i) \right|^2 + \lambda \left(\sum_{t_j} |u_\theta(x=0, t_j)|^2 + \sum_{t_j} |u_\theta(x=0, t_j)|^2 + \sum_{x_l} |u_\theta(x_l, t=0) - \sin(x_l)|^2 \right)$$

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2}, \qquad x \in]0, \pi[,\ t \in]0, 1],$$
$$u(x=0, t) = 0, \qquad t \in [0, 1],$$
$$u(x=\pi, t) = 0, \qquad t \in [0, 1],$$
$$u(x, t=0) = \sin(x), \qquad x \in [0, \pi].$$

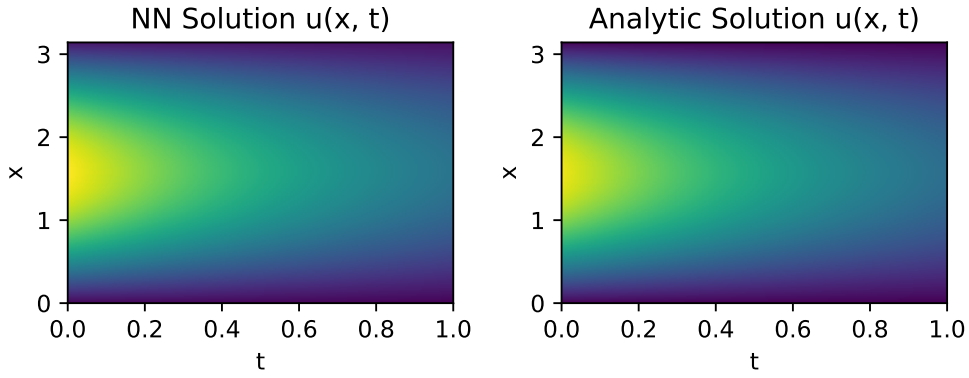
A minimal Example : Solving the Heat Equation

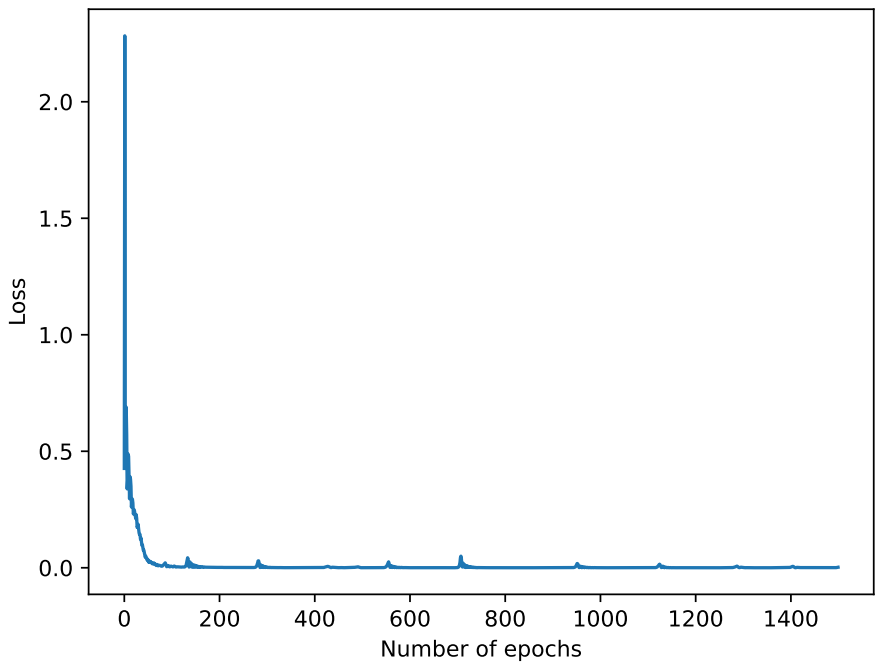
```python
import torch

x = torch.linspace(0, torch.pi, 50, requires_grad=True)
t = torch.linspace(0, 1, 50, requires_grad=True)
x, t = torch.meshgrid(x, t, indexing="ij")
x = x.reshape(-1, 1)
t = t.reshape(-1, 1)

mlp = torch.nn.Sequential(
    torch.nn.Linear(2, 10),
    torch.nn.Tanh(),
    torch.nn.Linear(10, 20),
    torch.nn.Tanh(),
    torch.nn.Linear(20, 1),
)
optimizer = torch.optim.Adam(list(mlp.parameters()), lr=0.05)

def dy_dx(y, x):
    return torch.autograd.grad(
        y, x, grad_outputs=torch.ones_like(y), create_graph=True
    )[0]

for i in range(1500):
    x = torch.rand(300, 1, requires_grad=True) * torch.pi  # Uniformly distributed values in [0, pi]
    t = torch.rand(300, 1, requires_grad=True)  # Uniformly distributed values in [0, 1]
```

A minimal Example : Solving the Heat Equation

```
for i in range(1500):
    x = torch.rand(300, 1, requires_grad=True) * torch.pi  # Uniformly distributed values in [0, pi]
    t = torch.rand(300, 1, requires_grad=True)  # Uniformly distributed values in [0, 1]

    # Concatenate x and t for input to the model
    inputs = torch.cat([x, t], dim=1)

    u = mlp(inputs)
    u_t = dy_dx(u, t)
    u_x = dy_dx(u, x)
    u_xx = dy_dx(u_x, x)

    # Boundary conditions
    residual_init = mlp(torch.cat([x, torch.zeros_like(t)], 1)) - torch.sin(x)
    residual_bdy1 = mlp(torch.cat([torch.zeros_like(x), t], 1))
    residual_bdy2 = mlp(torch.cat([torch.ones_like(x) * torch.pi, t], 1))

    # Loss calculation
    residual_pde = u_t - u_xx
    loss = ((residual_pde ** 2).mean() +
            (residual_init ** 2).mean() +
            (residual_bdy1 ** 2).mean() +
            (residual_bdy2 ** 2).mean())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

A minimal Example : Solving the Heat Equation





A note on computing derivatives with autograd

Naive autodiff scales **exponentially** in compute + memory in order of derivative.

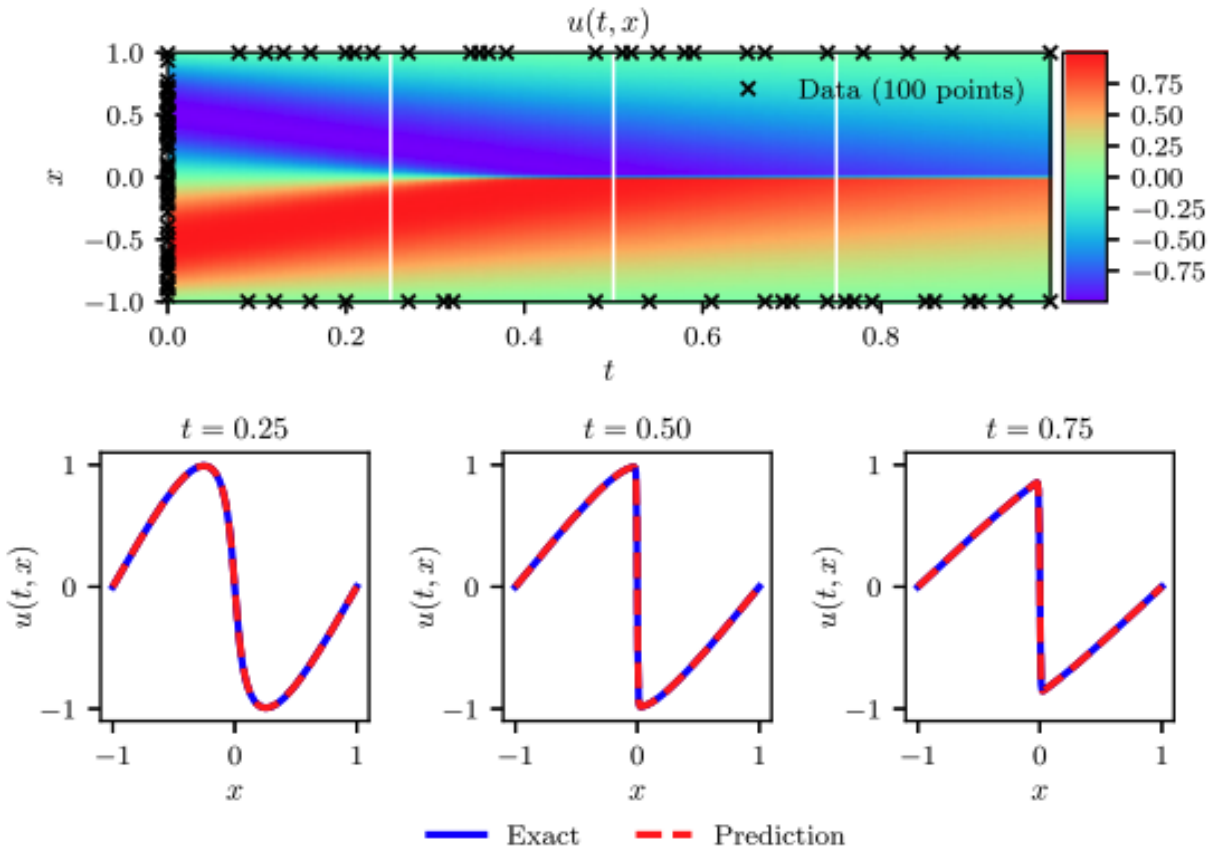
Faster alternatives e.g.

- Taylor Mode autodiff in *jax*,
- finite differences,
- Hutkinson Trace Estimation,

Burgers' Equation [Raissi et al., 2019]

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1],$$
$$u(0, x) = -\sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$

M. Raissi et al. / Journal of Computational Physics 378 (2019) 686–707

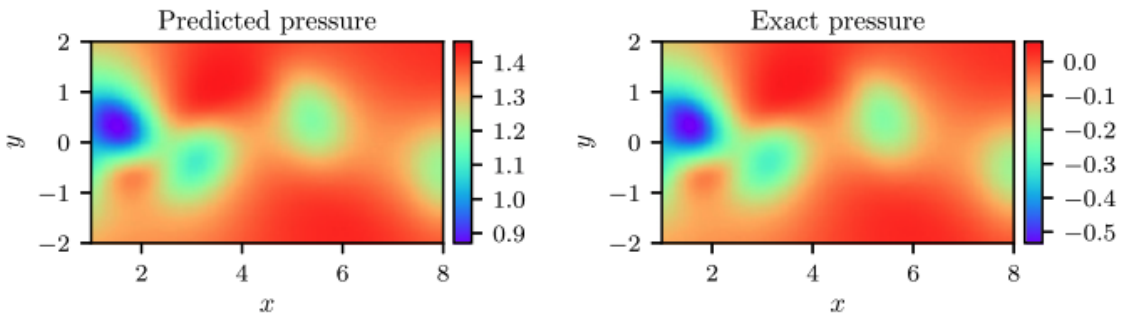


Inverse problems [Raissi et al., 2019]

Problem : Navier-Stokes equation, recovering parameters from data

$$u_t + \lambda_1(uu_x + vu_y) = -p_x + \lambda_2(u_{xx} + u_{yy}),$$
$$v_t + \lambda_1(uv_x + vv_y) = -p_y + \lambda_2(v_{xx} + v_{yy}),$$

Here, λ_1, λ_2 and p are the unknowns. 1% of the data used, added noise.



Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.999(uu_x + vu_y) = -p_x + 0.01047(u_{xx} + u_{yy})$ $v_t + 0.999(uv_x + vv_y) = -p_y + 0.01047(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$

Pros and Cons of Physics-Informed Neural Networks (PINNs)

Pros

- **Mesh-free**,
- solve forward and inverse problems jointly,
- Works for *messy* problems where observational data available,
- Tractable, analytical gradients,
- Does not require data,

Cons

- Computational cost,
- Hard to optimize in practice,
- May require significant tweeking,

Theoretical Analysis

Analysis

What we want to minimize :

$$\textbf{Closeness to Solution :} \quad \mathcal{E}(\theta) = \|u_\theta - u\|_p$$

What we would like to minimize :

$$\textbf{Quadrature Error :} \quad \mathcal{E}_G(\theta) = \|\mathcal{D}(u_\theta) - f\|_p$$

What we can compute :

$$\textbf{Training Error :} \quad \mathcal{E}_T(\theta) = \left(\sum_{i=1}^{N} |\mathcal{D}(u_\theta)(x_i) - f(x_i)|^p \right)^{\frac{1}{p}}$$

Questions :

1. Can we find u_θ such that residual $\mathcal{E}_G$ is small ?
2. Does a small Training Error $\mathcal{E}_T$ imply small error $\mathcal{E}$?

Question 1 : Can we find u_θ such that Residual $\mathcal{E}_G$ is small ?

$$\begin{aligned}\mathcal{E}_G(\theta) &:= \|\mathcal{D}(u_\theta) - f\|_p \\ &= \|\mathcal{D}(u_\theta) - \mathcal{D}(u)\|_p \\ &\leq \bar{C}_{\text{pde}}(u, u_\theta)\|u_\theta - u\|_{W^{s,p}}\end{aligned}$$

Universal Approximation Theorem (De Ryck et al. [2021])

$\exists\theta$ such that $\|u_\theta - u\|_{W^{s,p}} < \epsilon$

- So then given f is smooth enough, $\mathcal{E}_G(\theta) \leq \epsilon'$
- applies for linear PDEs,
- nonlinear PDEs where u and its derivatives are bounded [Ryck and Mishra, 2024],

Question 2 : Does a small Training Error $\mathcal{E}_T$ imply small $\mathcal{E}$?

Coercivity : $\|u_\theta - u\|_p \leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - \mathcal{D}(u)\|_p$

Coercivity bounds residual $\mathcal{E}_G(\theta)$:

$$\begin{aligned}\mathcal{E}(\theta) &\leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - \mathcal{D}(u)\|_p \\ &\leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - f\|_p \\ &\leq C_{\text{pde}}(u, u_\theta)\mathcal{E}_G(\theta)\end{aligned}$$

Bounds with Training error :

$$\mathcal{E}_G(\theta) \leq \mathcal{E}_T(\theta) + C_{\text{quad}}(u_{\theta^\star})^{\frac{1}{p}} N^{-\frac{\alpha}{p}}$$

Question 2 : Does a small Training Error $\mathcal{E}_T$ imply small $\mathcal{E}$?

Coercivity : $$\|u_\theta - u\|_p \leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - \mathcal{D}(u)\|_p$$

Coercivity bounds residual $\mathcal{E}_G(\theta)$:

$$\begin{aligned}\mathcal{E}(\theta) &\leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - \mathcal{D}(u)\|_p \\ &\leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - f\|_p \\ &\leq C_{\text{pde}}(u, u_\theta)\mathcal{E}_G(\theta)\end{aligned}$$

Bounds with Training error :

$$\mathcal{E}_G(\theta) \leq \mathcal{E}_T(\theta) + C_{\text{quad}}(u_{\theta^\star})^{\frac{1}{p}} N^{-\frac{\alpha}{p}}$$

- $\mathcal{E}(\theta)$ bounded by function of $\mathcal{E}_T(\theta)$
- Generic strategy to obtain bounds, applicable in many cases,

Question 2 : Does a small Training Error $\mathcal{E}_T$ imply small $\mathcal{E}$?

Coercivity : $$\|u_\theta - u\|_p \leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - \mathcal{D}(u)\|_p$$

Coercivity bounds residual $\mathcal{E}_G(\theta)$:

$$\begin{aligned}\mathcal{E}(\theta) &\leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - \mathcal{D}(u)\|_p \\ &\leq C_{\text{pde}}(u, u_\theta)\|\mathcal{D}(u_\theta) - f\|_p \\ &\leq C_{\text{pde}}(u, u_\theta)\mathcal{E}_G(\theta)\end{aligned}$$

Bounds with Training error :

$$\mathcal{E}_G(\theta) \leq \mathcal{E}_T(\theta) + C_{\text{quad}}(u_{\theta^\star})^{\frac{1}{p}} N^{-\frac{\alpha}{p}}$$

- $\mathcal{E}(\theta)$ bounded by function of $\mathcal{E}_T(\theta)$
- Generic strategy to obtain bounds, applicable in many cases,
- Caveats ? Not informative if :
  - PDE constant is large C_{pde},
  - n$^{\text{o}}$ collocation points small, or training err. large

Training Error ?

So far we have seen :

- There exists a network s.t. residual error is small,
- Small training residual implies small overall error,

But can $\mathcal{E}_T(\theta)$ be made small in practice ?

Difficulties in training PINNs

Example :

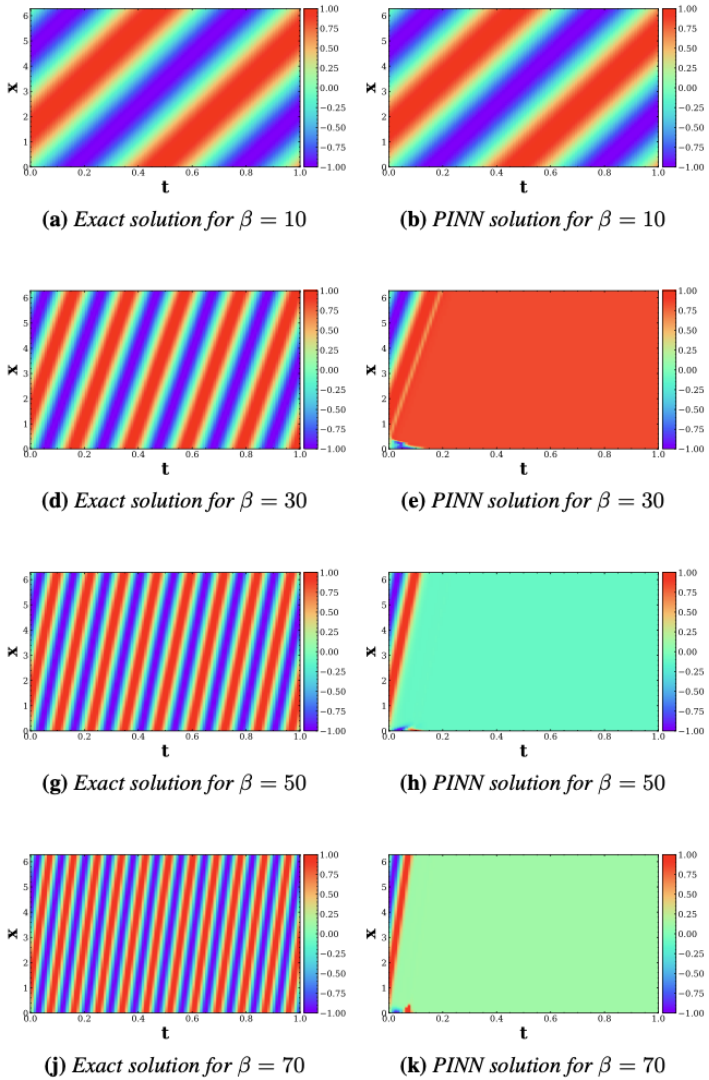
Linear transport equation

(on periodic domain)

$$u_t + \beta u_x = 0$$

For large β :

- Training difficulties
- Collapse to trivial solution



(a) *Exact solution for* $\beta = 10$ **(b)** *PINN solution for* $\beta = 10$

(d) *Exact solution for* $\beta = 30$ **(e)** *PINN solution for* $\beta = 30$

(g) *Exact solution for* $\beta = 50$ **(h)** *PINN solution for* $\beta = 50$

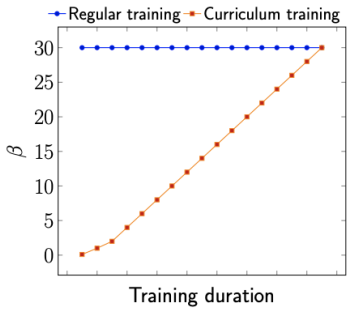
(j) *Exact solution for* $\beta = 70$ **(k)** *PINN solution for* $\beta = 70$

Krishnapriyan et al. (2021). *Characterizing possible failure modes in physics-informed neural networks*.

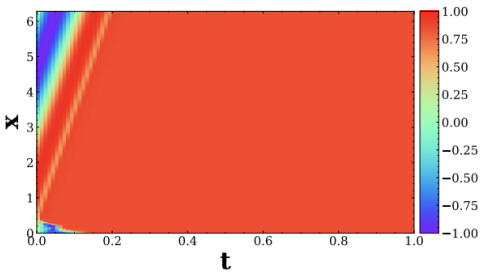
Tricks and techniques to get it to work

- Curriculum Learning,
- Domain decomposition, sequential learning,
- Fourier Features,
- Quasi-Second Order-like optimizers e.g. Adam, LBFGS
- More techniques here : Wang et al. [2023]

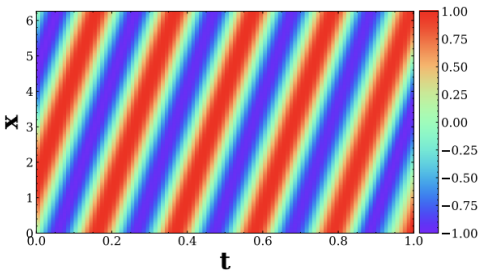
Curriculum Learning



(a) *Curriculum regularization schematic*

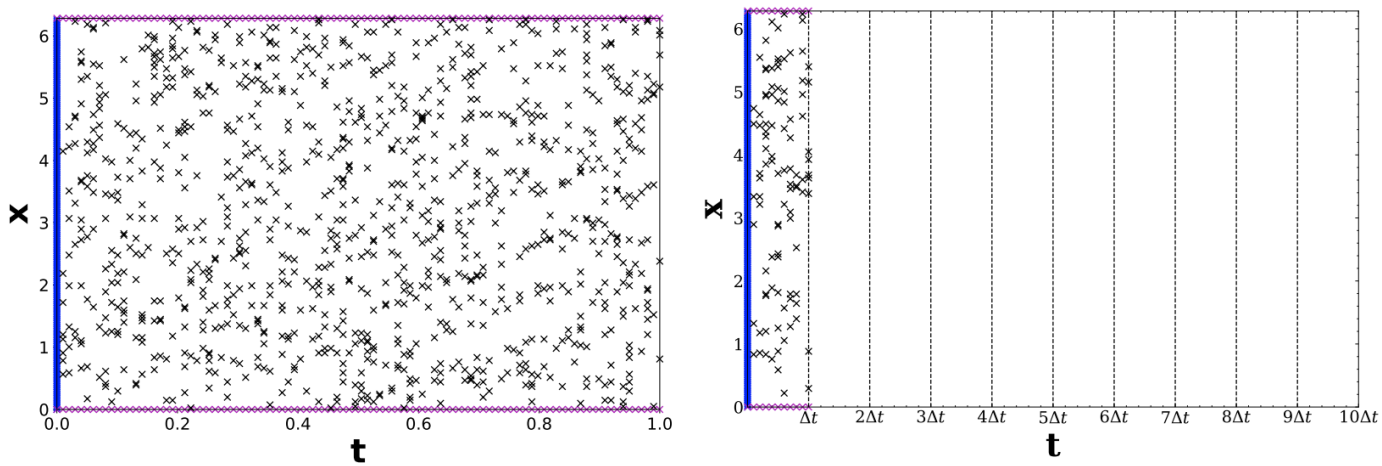


(b) *Regular training PINN solution for* $\beta = 30$



(c) *Curriculum training PINN solution for* $\beta = 30$

Sequential Learning



(a) *Regular PINN training*

(b) *Sequence-to-sequence learning (model trained every Δt)*

- Similar results to curriculum learning,

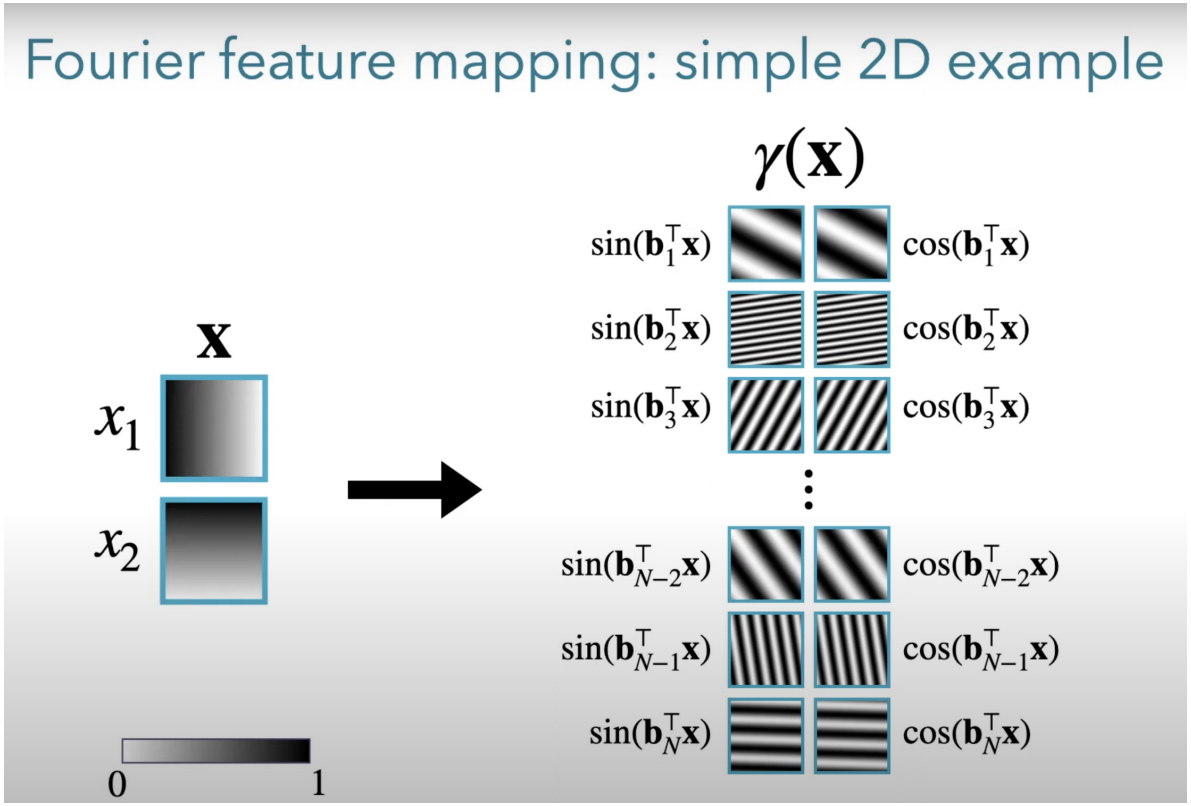


Figure 2 – From Tancik et al. [2020]

Fourier features

First layer transforms coordinates x :

$$\gamma(x) = [\cos(b_1^\top x), \sin(b_1^\top x), \ldots, \cos(b_n^\top x), \sin(b_n^\top x)], \quad b_j \sim \mathcal{N}$$

Fourier Features

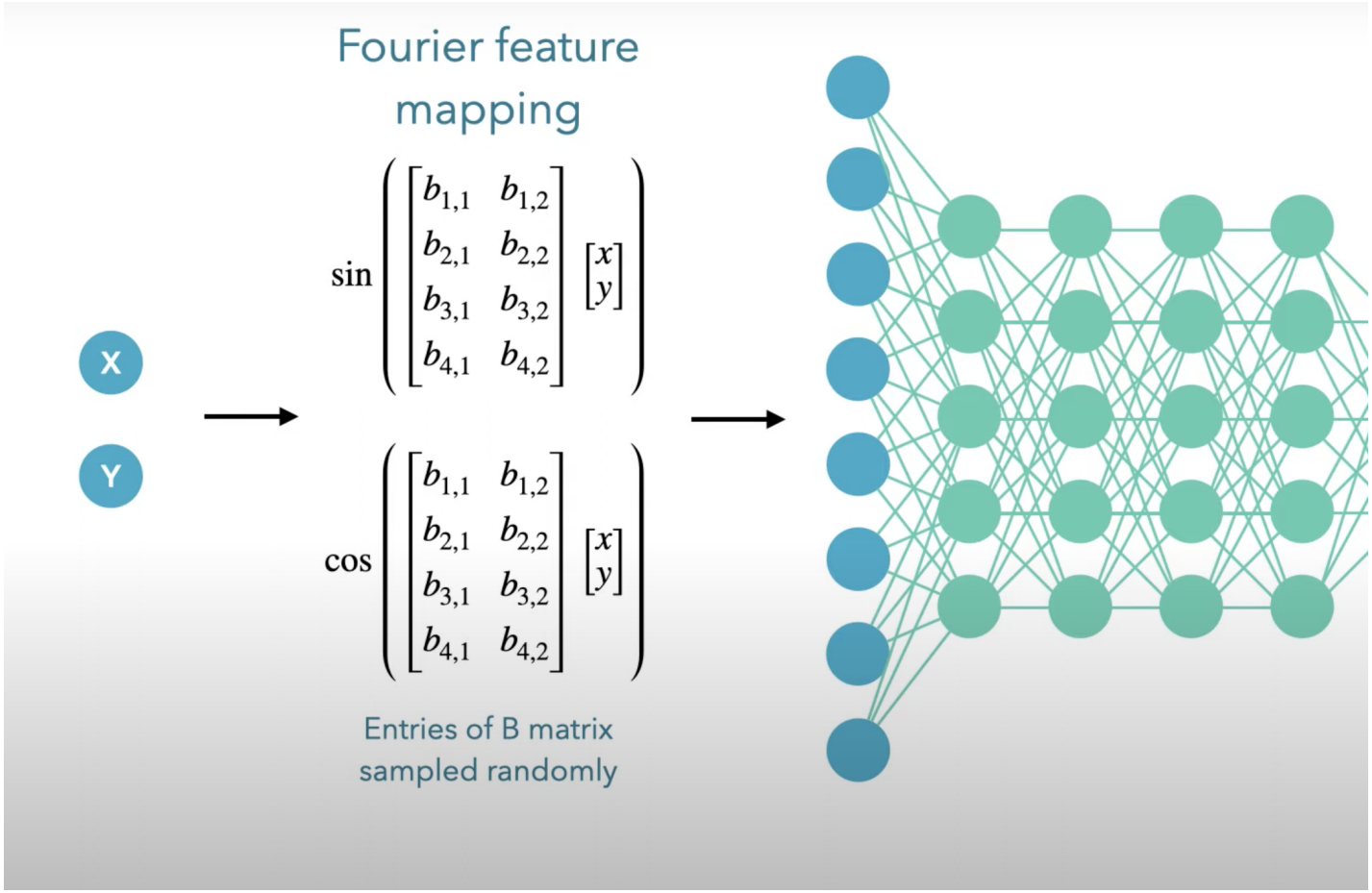


Figure 3 – From Tancik et al. [2020]

Fourier Features

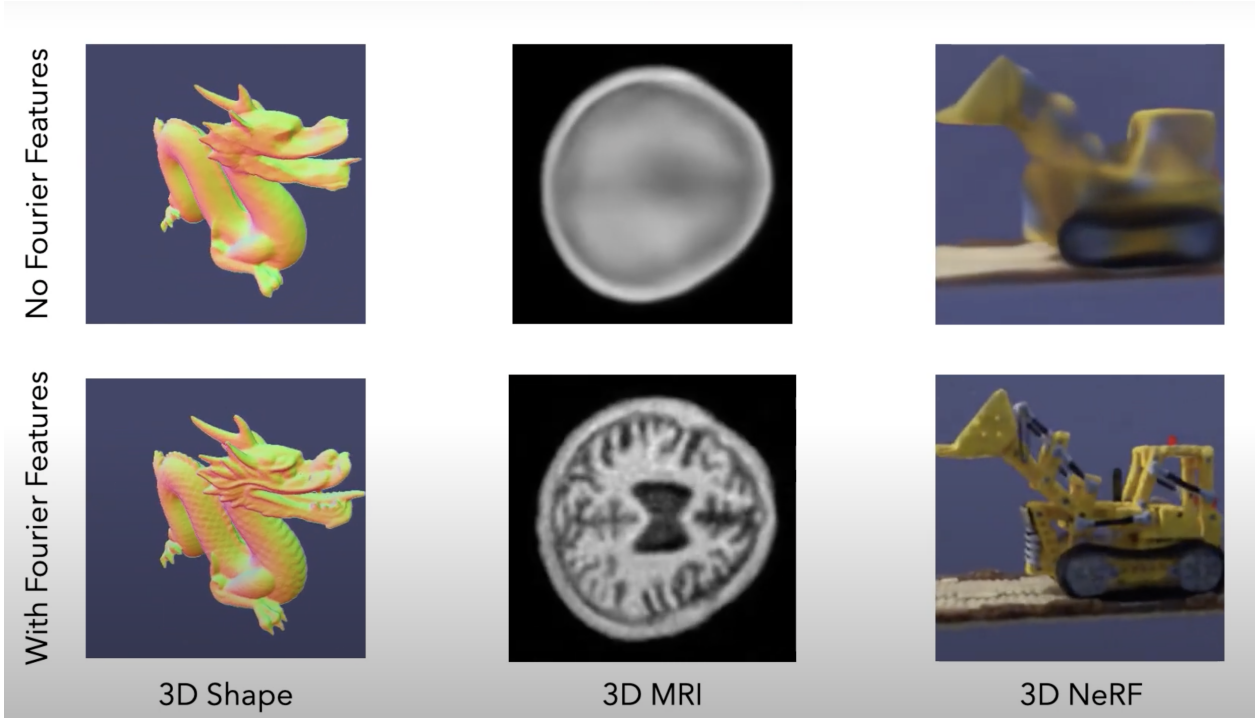


Figure 4 – From Tancik et al. [2020]

Fourier Features

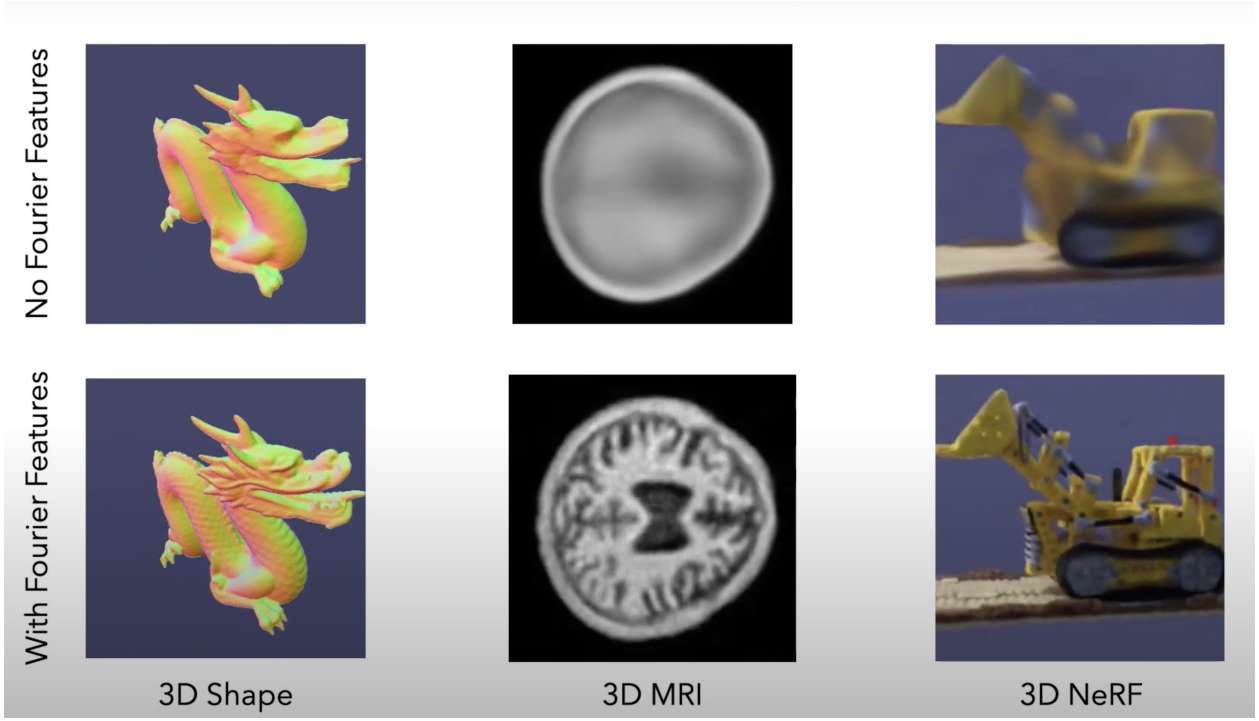


Figure 4 – From Tancik et al. [2020]

- All these tricks are not really understood,
- In order to understand these tricks & training errors we need to understand training,
- But analyzing NN training is very difficult...

Analysis of Training In a simplified setting [Ryck et al., 2024]

Assumptions

Linear PDE

- $\mathcal{D}(u) = f, \quad x \in \Omega$
- $\mathcal{B}(u) = 0, \quad x \in \partial\Omega, \qquad \mathcal{D}, \mathcal{B}$ **linear** operators

Simplified ansatz

- $u_\theta(x) = \sum_k \theta_k \phi_k(x)$
- can be extended to NNs

Will make training dynamics tractable,

- The following is informal, but made formal in [Ryck et al., 2024]

Poisson's Equation with zero BC

$$u''(x) = -k^2 \sin(kx),$$
$$u(-\pi) = 0,\ u(\pi) = 0. \tag{1}$$

The solution is given by $u(x) = \sin(kx)$.

$$\mathcal{L}_{\text{PDE}} = \mathcal{L}_{\textbf{Res}} + \lambda \mathcal{L}_{\text{BC}}, \quad \mathcal{L}_{\textbf{Res}} = \int_{\Omega} \left| u_\theta''(x) + k^2 \sin(kx) \right|^2 dx,$$

$$\mathcal{L}_{\text{BC}} = \frac{1}{2} \left[u_\theta(-\pi)^2 + u_\theta(\pi)^2 \right].$$

Fourier Features Ansatz $u_\theta : \Omega \mapsto \mathbb{R}$

$$u_\theta(x) = \sum_{k=-K}^{K} \theta_k \phi_k(x), \quad \phi_0(x) = \frac{1}{\sqrt{2\pi}},$$

$$\phi_{-k}(x) = \frac{1}{\sqrt{\pi}} \cos(kx), \quad \phi_k(x) = \frac{1}{\sqrt{\pi}} \sin(kx) \quad \text{for } 1 \leq k \leq K.$$

Gradient Descent Equations

$$\begin{aligned} \theta_{l+1} &= \theta_l - \eta \nabla_\theta \mathcal{L}(\theta_l) \\ &= \theta_l - \eta \nabla_\theta (\mathcal{L}_{\textbf{Res}} + \lambda \mathcal{L}_{\textbf{BC}})(\theta_l) \end{aligned}$$

Working Example : Calculating the Gradient

$$
\begin{aligned}
\nabla_\theta \mathcal{L}_{\mathbf{Res}} &= \nabla_\theta \frac{1}{2} \int_\Omega |\mathcal{D}(u_\theta)(x) - f(x)|^2 \, dx \\
&= \int_\Omega \nabla_\theta u_\theta(x) \mathcal{D}^*(\mathcal{D}(u_\theta)(x) - f(x)) \, dx \\
&= \int_\Omega \overbrace{\nabla_\theta u_\theta(x)}^{\phi(x) := (\phi_{-K}(x), \ldots, \phi_K(x))^\top} \mathcal{D}^*\mathcal{D}(\overbrace{u_\theta}^{\phi(x)^\top \cdot \theta})(x) - \underbrace{\phi(x)^\top \mathcal{D}^* f(x)}_{\text{indep. of } \theta} \, dx \\
&= \int_\Omega \phi(x) \mathcal{D}^*\mathcal{D}\phi(x)^\top \theta \, dx + \text{constant} \\
&= \underbrace{\left(\int_\Omega \phi(x) \overbrace{\mathcal{D}^*\mathcal{D}}^{= \frac{d^{\mathbf{4}}}{dx^{\mathbf{4}}}} \phi(x)^\top dx \right)}_{=: A_{\mathbf{Res}},\ \text{an } (2K+1)\times(2K+1) \text{ matrix}} \cdot \underbrace{\theta}_{\text{vector of size } 2K+1} + \text{constant} \\
&= A_{\text{Res}} \cdot \theta + \text{constant}
\end{aligned}
$$

Working Example : Calculating the Gradient

$$u_\theta(x) = \sum_{k=-K}^{K} \theta_k \phi_k(x)$$

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\mathbf{Res}} &= \nabla_\theta \frac{1}{2} \int_\Omega |\mathcal{D}(\iota \\
&= \int_\Omega \nabla_\theta u_\theta(x) \mathcal{1} \\
&= \int_\Omega \overbrace{\nabla_\theta u_\theta(x)}^{\phi(x) := (\phi_{-K}(x), \ldots, \phi_K(x))^\top} \mathcal{D}^*\mathcal{D}(\overbrace{u_\theta}^{\phi(x)^\top \cdot \theta})(x) - \underbrace{\phi(x)^\top \mathcal{D}^* f(x)}_{\text{indep. of } \theta} \, dx \\
&= \int_\Omega \phi(x) \mathcal{D}^*\mathcal{D}\phi(x)^\top \theta dx + \text{constant} \\
&= \underbrace{\left(\int_\Omega \phi(x) \overbrace{\mathcal{D}^*\mathcal{D}}^{= \frac{d^{\mathbf{4}}}{dx^{\mathbf{4}}}} \phi(x)^\top dx \right)}_{=: A_{\mathbf{Res}},\ \text{an } (2K+1)\times(2K+1) \text{ matrix}} \cdot \underbrace{\theta}_{\text{vector of size } 2K+1} + \text{constant} \\
&= A_{\text{Res}} \cdot \theta + \text{constant}
\end{aligned}$$

Working Example : Calc

$$\frac{d^4}{dx^4}\cos(kx) = k^4 cos(kx)$$

$$\nabla_\theta \mathcal{L}_{\text{Res}} = \nabla_\theta \frac{1}{2}\int_\Omega |\mathcal{D}(\iota$$

$$= \int_\Omega \nabla_\theta u_\theta(x) 1$$

$$= \int_\Omega \overbrace{\nabla_\theta}^{\phi(x):=(\phi_{-K}} $$

$$= \int_\Omega \phi(x)\mathcal{D}^*\mathcal{D}$$

$$= \underbrace{\left(\int_\Omega \phi(x) \overbrace{\mathcal{D}^*\mathcal{D}}^{=\frac{d^4}{dx^4}} \phi(x)^\top dx\right)}_{=: A_{\text{Res}},\ \text{an } (2K+1)\times(2K+1)\ \text{ matrix}} \cdot \underbrace{\theta}_{\text{vector of size } 2K+1} + \text{constant}$$

$$= A_{\text{Res}} \cdot \theta + \text{constant}$$

$$A_{\text{Res}} = \begin{bmatrix} (-K)^4 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \cdots & 0 & 0 \\ 0 & \cdots & 0^4 & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & K^4 \end{bmatrix}$$

Computing the Gradient

$$\nabla_\theta \mathcal{L}_{\mathrm{BC}}(\theta) = \overbrace{\phi(x)\phi(x)^\top}^{=:A_{\mathbf{BC}},\ \text{rank-1}\ \ (2K+1)\times(2K+1)\ \ \text{matrix}} \cdot\theta = A_{\mathrm{BC}} \cdot \theta$$

So :

$$\nabla_\theta(\mathcal{L}(\theta_l) = (\overbrace{A_{\mathrm{Res}} + \lambda A_{\mathrm{BC}}}^{=:A})\theta + \text{constant}$$

$$A = \begin{bmatrix} (-K)^4 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \cdots & 0 & 0 \\ 0 & \cdots & 0^4 & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & K^4 \end{bmatrix} + \lambda \begin{bmatrix} \phi_{-K}(\pi) \\ \vdots \\ \phi_0(\pi) \\ \vdots \\ \phi_K(\pi) \end{bmatrix} \begin{bmatrix} \phi_{-K}(\pi), & \cdots, & \phi_K(\pi) \end{bmatrix}$$

Gradient Descent Equations

$$\theta_{l+1} = \theta_l - \eta \nabla_\theta (\mathcal{L}_{\mathbf{Res}} + \lambda \mathcal{L}_{\mathbf{BC}})(\theta_l) \quad = \left(I - \eta \underbrace{A}_{A_{\mathbf{Res}} + \lambda A_{\mathbf{BC}}} \right) \theta_l + \text{constant}$$

Convergence of Training

Gradient Descent Equations

$$\theta_{l+1} = \theta_l - \eta \nabla_\theta (\mathcal{L}_{\mathbf{Res}} + \lambda \mathcal{L}_{\mathbf{BC}})(\theta_l) \quad = \left(I - \eta \underbrace{A}_{A_{\mathbf{Res}} + \lambda A_{\mathbf{BC}}} \right) \theta_l + \text{constant}$$

Convergence rate

Let $0 < c < 1$ and let A be invertible with condition number $\kappa(A)$,

$$\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A). \tag{2}$$

Set $\eta = c/\lambda_{\max}(A)$. Then,

$$\|\theta_l - \theta^*\|_2 \leq (1 - c/\kappa(A))^l \, \|\theta_0 - \theta^*\|_2. \tag{3}$$

Convergence of Training

Gradient Descent Equations

$$\theta_{l+1} = \theta_l - \eta \nabla_\theta (\mathcal{L}_{\mathbf{Res}} + \lambda \mathcal{L}_{\mathbf{BC}})(\theta_l) \quad = \left(I - \eta \underbrace{A}_{A_{\mathbf{Res}} + \lambda A_{\mathbf{BC}}} \right) \theta_l + \text{constant}$$

Convergence rate

Let $0 < c < 1$ and let A be invertible with condition number $\kappa(A)$,

$$\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A). \tag{2}$$

Set $\eta = c/\lambda_{\max}(A)$. Then,

$$\|\theta_l - \theta^*\|_2 \leq (1 - c/\kappa(A))^l \, \|\theta_0 - \theta^*\|_2. \tag{3}$$

Convergence of Training

Gradient Descent Equations

$$\theta_{l+1} = \theta_l - \eta \nabla_\theta (\mathcal{L}_{\mathbf{Res}} + \lambda \mathcal{L}_{\mathbf{BC}})(\theta_l) \quad = \left(I - \eta \underbrace{A}_{A_{\mathbf{Res}} + \lambda A_{\mathbf{BC}}} \right) \theta_l + \text{constant}$$

Convergence rate

Let $0 < c < 1$ and let A be invertible with condition number $\kappa(A)$,

$$\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A). \tag{2}$$

Set $\eta = c/\lambda_{\max}(A)$. Then,

$$\|\theta_l - \theta^*\|_2 \leq (1 - c/\kappa(A))^l \, \|\theta_0 - \theta^*\|_2. \tag{3}$$

Corollary

For $\|\theta_l - \theta^*\|_2 \leq \varepsilon$ you need $O\left(\kappa(A) \ln \frac{1}{\epsilon}\right)$ GD steps .

Convergence rate, Working example

$$A = \begin{bmatrix} (-K)^4 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \cdots & 0 & 0 \\ 0 & \cdots & 0^4 & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & K^4 \end{bmatrix} + \lambda \begin{bmatrix} \phi_{-K}(\pi) \\ \vdots \\ \phi_0(\pi) \\ \vdots \\ \phi_K(\pi) \end{bmatrix} \begin{bmatrix} \phi_{-K}(\pi), & \cdots, & \phi_K(\pi) \end{bmatrix}$$

Convergence rate, Working example

$$A = \begin{bmatrix} (-K)^4 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \cdots & 0 & 0 \\ 0 & \cdots & 0^4 & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & K^4 \end{bmatrix} + \lambda \begin{bmatrix} \phi_{-K}(\pi) \\ \vdots \\ \phi_0(\pi) \\ \vdots \\ \phi_K(\pi) \end{bmatrix} \begin{bmatrix} \phi_{-K}(\pi), & \cdots, & \phi_K(\pi) \end{bmatrix}$$

- In the zero-BC Poisson case, $\kappa(A) \geq K^4$,
- if 500 steps required for $K = 5$, roughly, 312 500 steps for $K = 25$.
- this is only for Poisson.. what about other (linear) PDEs ?

The general Linear setting

Linear PDE

- $\mathcal{D}(u) = f, \quad x \in \Omega$
- $\mathcal{B}(u) = 0, \quad x \in \partial\Omega, \qquad \mathcal{D}, \mathcal{B}$ **linear** operators

Omitting boundary loss for simplicity (studied in paper), GD dynamics :

$$\theta_{l+1} = \theta_l - \eta \nabla_\theta \mathcal{L} = \Big(1 - \eta \overbrace{\qquad\qquad}^{\approx \int_\Omega \phi(x) \mathcal{D}^* \mathcal{D} \phi(x)^\top dx} A \qquad\qquad \Big) \cdot \theta_l + \text{constant}$$

The general Linear setting

Linear PDE

- $\mathcal{D}(u) = f, \quad x \in \Omega$
- $\mathcal{B}(u) = 0, \quad x \in \partial\Omega, \qquad \mathcal{D}, \mathcal{B}$ **linear** operators

Omitting boundary loss for simplicity (studied in paper), GD dynamics :

$$\theta_{l+1} = \theta_l - \eta \nabla_\theta \mathcal{L} = \Big(1 - \eta \overbrace{\qquad A \qquad}^{\approx \int_\Omega \phi(x)\mathcal{D}^*\mathcal{D}\phi(x)^\top dx} \Big) \cdot \theta_l + \text{constant}$$

Operator Perspective

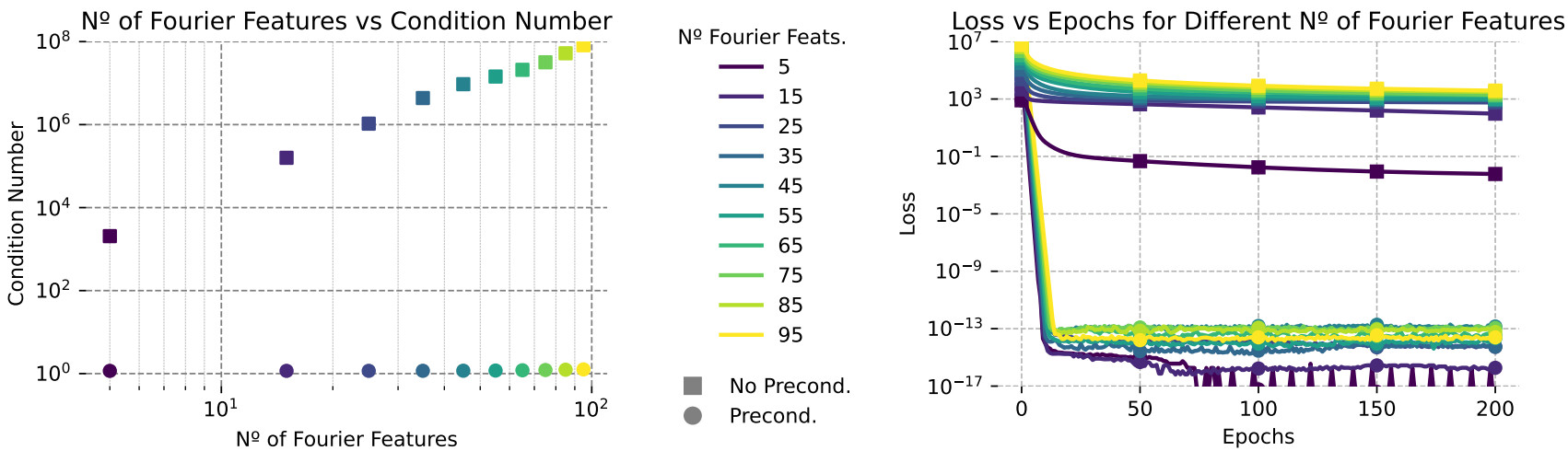
$$\kappa\Big(\int_\Omega \phi(x)\mathcal{D}^*\mathcal{D}\phi(x)^\top dx\Big) \qquad \longleftrightarrow \qquad \kappa(\mathcal{D}^*\mathcal{D} \circ \mathscr{T})$$

- Hermitian-squared operator $\mathcal{D}^*\mathcal{D}$, e.g. for Poisson $\Delta^2 = \Delta \circ \Delta$
- Kernel integral operator $\mathscr{T}[f](x) := \displaystyle\int f(y)\phi(x)^\top \phi(y) dy$
- In particular, $\mathscr{T} = \mathrm{Id}$ if $\{\phi_k\}_k$ orthonormal basis (if input in span)
- conditioning essentially governed by conditioning of $\mathcal{D}^*\mathcal{D}$

Back to the Poisson equation Example

PDE :

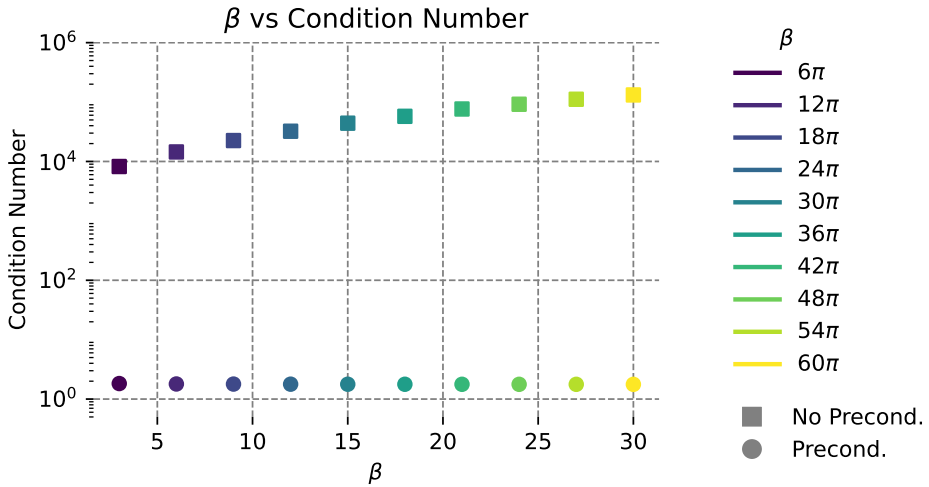
$$u''(x) = -k^2 \sin(kx),$$
$$u(-\pi) = 0,\ u(\pi) = 0.$$

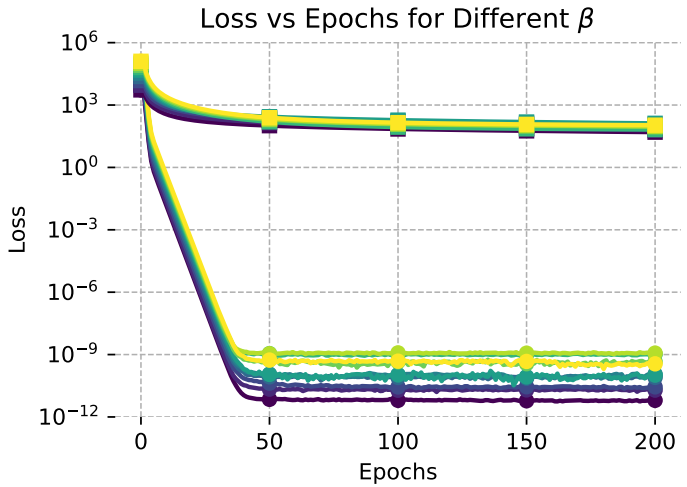


Observation : $\kappa(A) \sim K^4$

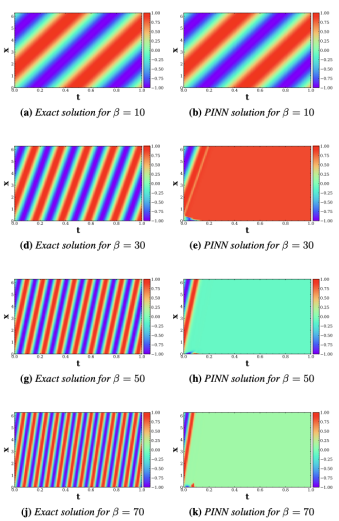
Linear advection equation with Fourier basis model

PDE : $u_t + \beta u_x = 0$ on $[0, T] \times [0, 1]$ with $u_0 = \sin$ and periodic BC





Observation : $\kappa(A) \sim \beta^2$



What about neural networks ?

So far the theory holds for the ansatz $u_\theta(x) = \sum_{k=-K}^{K} \theta_k \phi_k(x)$

Idea : Taylor expand the training dynamics of the NN :

The so called Neural Tangent Kernel (NTK) regime [Jacot et al., 2018] :

$$u(x;\theta_k) \approx u(x;\theta_0) + \nabla_\theta u(x;\theta_0)^\top (\theta_k - \theta_0) \tag{4}$$

Result : Assuming $\mathrm{Hess}[\mathcal{L}(\theta_k)]$ and $\mathcal{D}\mathrm{Hess}[\mathcal{L}(\theta_k)]$ small enough :

$$\theta_{l+1} = (I - \eta A)\,\theta_l + \text{constant}$$

Same exact results as in linear basis setting, with $\phi(x) = \nabla_\theta u(x;\theta_0)$

What about neural networks ?

So far the theory holds for the ansatz $u_\theta(x) = \sum_{k=-K}^{K} \theta_k \phi_k(x)$

Idea : Taylor expand the training dynamics of the NN :

The so called Neural Tangent Kernel (NTK) regime [Jacot et al., 2018] :

$$u(x;\theta_k) \approx u(x;\theta_0) + \nabla_\theta u(x;\theta_0)^\top (\theta_k - \theta_0) \tag{4}$$

Result : Assuming $\mathrm{Hess}[\mathcal{L}(\theta_k)]$ and $\mathcal{D}\mathrm{Hess}[\mathcal{L}(\theta_k)]$ small enough :

$$\theta_{l+1} = (I - \eta A)\,\theta_l + \text{constant}$$

Same exact results as in linear basis setting, with $\phi(x) = \nabla_\theta u(x;\theta_0)$

- Assumption holds exactly when
  - linear basis,
  - neural network, when width grows unbounded (in the NTK regime),

What about neural networks ?

So far the theory holds for the ansatz $u_\theta(x) = \sum_{k=-K}^{K} \theta_k \phi_k(x)$

Idea : Taylor expand the training dynamics of the NN :

The so called Neural Tangent Kernel (NTK) regime [Jacot et al., 2018] :

$$u(x;\theta_k) \approx u(x;\theta_0) + \nabla_\theta u(x;\theta_0)^\top (\theta_k - \theta_0) \tag{4}$$

Result : Assuming $\mathrm{Hess}[\mathcal{L}(\theta_k)]$ and $\mathcal{D}\mathrm{Hess}[\mathcal{L}(\theta_k)]$ small enough :

$$\theta_{l+1} = (I - \eta A)\,\theta_l + \text{constant}$$

Same exact results as in linear basis setting, with $\phi(x) = \nabla_\theta u(x;\theta_0)$

- Assumption holds exactly when
  - linear basis,
  - neural network, when width grows unbounded (in the NTK regime),
- May not hold in practice. But yields insights into training difficulties,

Operator Perspective

$$\kappa\Big(\int_\Omega \phi(x)\mathcal{D}^*\mathcal{D}\phi(x)^\top dx\Big) \qquad \longleftrightarrow \qquad \kappa(\mathcal{D}^*\mathcal{D} \circ \mathscr{T})$$

- $\mathscr{T}[f](x) := \int f(y)\phi(x)^\top\phi(y)dy$ encodes the architecture,
- $\phi(x)^\top\phi(x) = \nabla_\theta u(x;\theta_0)^\top \nabla_\theta u(x;\theta_0)$ is the NTK

Operator Perspective

$$\kappa\Big(\int_\Omega \phi(x)\mathcal{D}^*\mathcal{D}\phi(x)^\top dx\Big) \qquad \longleftrightarrow \qquad \kappa(\mathcal{D}^*\mathcal{D} \circ \mathscr{T})$$

- $\mathscr{T}[f](x) := \int f(y)\phi(x)^\top\phi(y)dy$ encodes the architecture,
- $\phi(x)^\top\phi(x) = \nabla_\theta u(x;\theta_0)^\top \nabla_\theta u(x;\theta_0)$ is the NTK

Spectrum of $\mathscr{T}[f]$ decays very rapidly in Fourier [Basri et al., 2020] :

- for uniform data on hypersphere $\mathbb{S}^{d-1}$,
  for residual ReLU networks, the NTK has :
- eigenfunctions = spherical harmonics
- eigenvalues decay polynomially with frequency k^{-d}

What about neural networks ?

Operator Perspective

$$\kappa\Big(\int_\Omega \phi(x)\mathcal{D}^*\mathcal{D}\phi(x)^\top dx\Big) \qquad \longleftrightarrow \qquad \kappa(\mathcal{D}^*\mathcal{D} \circ \mathscr{T})$$

- $\mathscr{T}[f](x) := \int f(y)\phi(x)^\top\phi(y)dy$ encodes the architecture,
- $\phi(x)^\top\phi(x) = \nabla_\theta u(x;\theta_0)^\top \nabla_\theta u(x;\theta_0)$ is the NTK

Spectrum of $\mathscr{T}[f]$ decays very rapidly in Fourier [Basri et al., 2020] :

- for uniform data on hypersphere $\mathbb{S}^{d-1}$,
  for residual ReLU networks, the NTK has :
- eigenfunctions = spherical harmonics
- eigenvalues decay polynomially with frequency k^{-d}

In contrast with regression, convergence given by $\kappa(\mathscr{T})$

Preconditioning to improve training

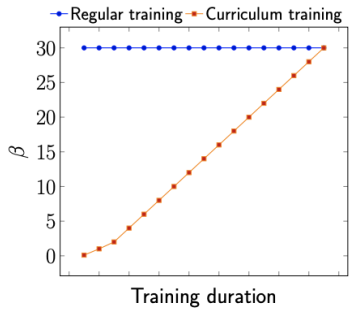
General idea : choose model u_θ (and hence $\mathscr{T}$) such that

$$\mathscr{T} \approx (\mathcal{D}^*\mathcal{D})^{-1} \quad \text{and hence} \quad \kappa(\mathcal{D}^*\mathcal{D} \circ \mathscr{T}) \approx 1$$

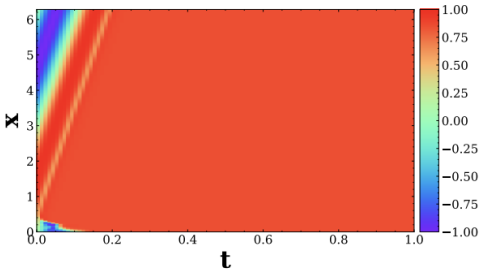
Revisiting Tricks and Techniques to get it to work

- Curriculum Learning,
- Domain decomposition, sequential learning,
- Fourier Features,
- Quasi-Second Order-like optimizers e.g. Adam, LBFGS
- More techniques here : Wang et al. [2023]

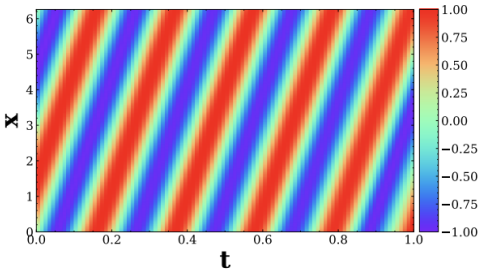
Curriculum Learning



(a) *Curriculum regularization schematic*



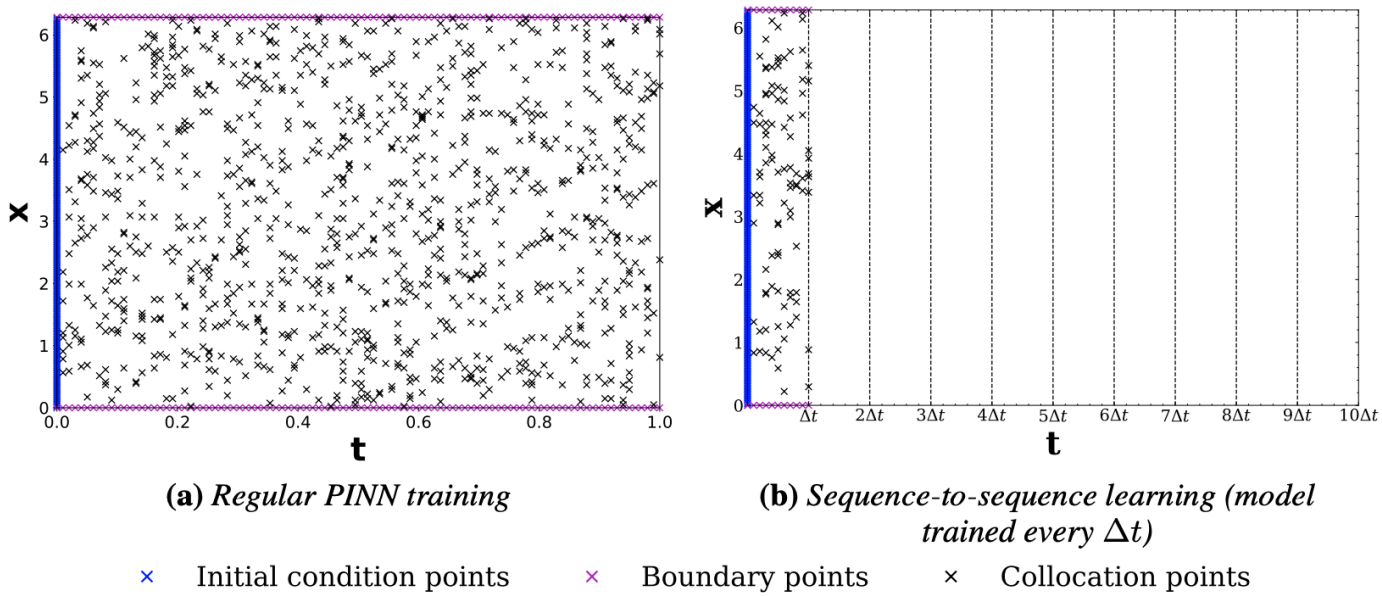
(b) *Regular training PINN solution for* $\beta = 30$



(c) *Curriculum training PINN solution for* $\beta = 30$

- $\kappa(A) \sim \beta^2$

Sequence to Sequence Learning



(a) *Regular PINN training*

(b) *Sequence-to-sequence learning (model trained every Δt)*

- Split $[0, T]$ into N subintervals leads to $\kappa(A) \sim \beta^2/N^2$

Second-order optimizers

Newton's method is given by

$$\theta_{k+1} = \theta_k - \eta \mathrm{Hess}[L(\theta_k)]^{-1} \nabla_\theta L(\theta_k)$$

For linear models :

$$\mathrm{Hess}[L(\theta_k)] = A$$

Newton's method = gradient preconditioning using A

- but costly...

Fourier Features [Tancik et al., 2020]

Recall the operator of interest is $\mathcal{D}^*\mathcal{D} \circ \mathscr{T}$, $\quad \mathscr{T}[f] := \int f(y)k(\cdot, y)dy$.

Spectrum of $\mathscr{T}[f]$ decays very rapidly in Fourier [Basri et al., 2020].

Fourier features

First layer transforms coordinates x :

$$\gamma(x) = [\cos(b_1 x), \sin(b_1 x), \ldots, \cos(b_n x), \sin(b_n x)], \quad b_j \sim \mathcal{N}$$

In this case :

$$k(x, y) := \nabla_\theta u_\theta(x)^\top \nabla_\theta u_\theta(y) = f \circ h(x - y)$$

$$h(x) := \sum_{j=1}^{m} \cos(2\pi b_j^\top (x - y))$$

- in the 1 hidden layer case, $\mathscr{T}$ is a convolution with kernel h,
- adjusts frequencies based on $\mathbb{P}(b)$
- similarity with Dirichlet kernel (if $b_j = j$),

To Conclude

- We have seen PINNs, with some tips and tricks
- PINNs have useful theoretical guarantees for quadrature + approximation errors,
- Training error is usually the hardest to minimize,
- Training problem severly ill-conditioned,
- **Preconditioning** PINNs ?
- No general recipe ? it is **PDE dependant**. Leverage tools from numerical analysis ?

To Conclude

- We have seen PINNs, with some tips and tricks
- PINNs have useful theoretical guarantees for quadrature + approximation errors,
- Training error is usually the hardest to minimize,
- Training problem severly ill-conditioned,
- **Preconditioning** PINNs ?
- No general recipe ? it is **PDE dependant**. Leverage tools from numerical analysis ?

• Deep learning is somewhat magical, learning problem highly non-convex. Doesn't mean that deep learning can solve any non-convex problem !

To Conclude

- We have seen PINNs, with some tips and tricks
- PINNs have useful theoretical guarantees for quadrature + approximation errors,
- Training error is usually the hardest to minimize,
- Training problem severly ill-conditioned,
- **Preconditioning** PINNs ?
- No general recipe ? it is **PDE dependant**. Leverage tools from numerical analysis ?

• Deep learning is somewhat magical, learning problem highly non-convex. Doesn't mean that deep learning can solve any non-convex problem !

• Interesting results in high dimensions.

References

References I

Ronen Basri, Meirav Galun, Amnon Geifman, David W. Jacobs, Yoni Kasten, and Shira Kritchman. Frequency bias in neural networks for input of non-uniform density. *CoRR*, abs/2003.04560, 2020. URL `https://arxiv.org/abs/2003.04560`.

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maizar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks : Where we are and what's next. *CoRR*, abs/2201.05624, 2022. URL `https://arxiv.org/abs/2201.05624`.

Tim De Ryck, Samuel Lanthaler, and Siddhartha Mishra. On the approximation of functions by tanh neural networks. *Neural Networks*, 2021. ISSN 0893-6080. doi : https://doi.org/10.1016/j.neunet.2021.08.015. URL `https://www.sciencedirect.com/science/article/pii/S0893608021003208`.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel : Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

References II

I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5) :987–1000, 1998. doi : $10.1109/72.712178$.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks : A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378 :686–707, 2019.

Tim De Ryck and Siddhartha Mishra. Numerical analysis of physics-informed neural networks and related models in physics-informed machine learning, 2024.

Tim De Ryck, Florent Bonnet, Siddhartha Mishra, and Emmanuel de Bézenac. An operator preconditioning perspective on training in physics-informed machine learning, 2024.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33 :7537–7547, 2020.

References III

Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An expert's guide to training physics-informed neural networks, 2023.

Analysis of GD

We want to analyse GD :

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta L(\theta_k)$$

By **Taylor's theorem** it holds that

$$u(x; \theta_k) = u(x; \theta_0) + \nabla_\theta u(x; \theta_0)^\top (\theta_k - \theta_0) + \epsilon_k$$

To do : prove formula for GD s.t. 2nd order term can be neglected

In the following : assume that $\mathcal{D}$ is linear

Analysis of GD

Define $\phi_i(x) = \partial_{\theta_i} u(x;\theta_0)$ and

$$A_{i,j} = \langle \mathcal{D}\phi_i, \mathcal{D}\phi_j \rangle_{L^2(\Omega)} + \lambda \langle \phi_i, \phi_j \rangle_{L^2(\partial\Omega)}$$
$$B_i = \langle f - \mathcal{D}u_{\theta_0}, \mathcal{D}\phi_i \rangle_{L^2(\Omega)} + \lambda \langle u - u_{\theta_0}, \phi_i \rangle_{L^2(\partial\Omega)}$$

then GD is given by

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta L(\theta_k) = (I - \eta A)\theta_k + \eta(A\theta_0 + B) + \eta\epsilon_k$$

where ϵ_k depends on H_k and $\mathcal{D}H_k$. Now let

$$\tilde{\theta}_{k+1} = (I - \eta A)\tilde{\theta}_k + \eta(A\tilde{\theta}_0 + B), \quad \tilde{\theta}_0 = \theta_0$$

Lemma

If η small and A invertible then it holds for any $k \in \mathbb{N}$ that,

$$\|\theta_k - \tilde{\theta}_k\|_2 \leq \max_{\ell \leq k} \|\epsilon_\ell\|_2 / \min_j |\lambda_j(A)|.$$

Analysis of GD

So far : $\theta_k \approx \tilde{\theta}_k \quad \Leftarrow \quad \epsilon_k \approx 0 \quad \Longleftrightarrow \quad H_k \approx 0$ and $\mathcal{D}H_k \approx 0$

Analysis of GD

So far : $\theta_k \approx \tilde{\theta}_k \quad \Leftarrow \quad \epsilon_k \approx 0 \quad \Longleftrightarrow \quad H_k \approx 0$ and $\mathcal{D}H_k \approx 0$

Linear u_θ : $\epsilon_k = 0$ so $\theta_k = \tilde{\theta}_k$

Analysis of GD

So far : $\theta_k \approx \tilde{\theta}_k \quad \Leftarrow \quad \epsilon_k \approx 0 \quad \Longleftrightarrow \quad H_k \approx 0$ and $\mathcal{D}H_k \approx 0$

Linear u_θ : $\epsilon_k = 0$ so $\theta_k = \tilde{\theta}_k$

Nonlinear u_θ :

approx. linearity $\Longleftrightarrow$ approx. constancy of (neural) tangent kernel,

$$\Theta[u_\theta](x, y) := \nabla_\theta u_\theta(x)^\top \nabla_\theta u_\theta(y)$$

Liu, Zhu, Belkin (2020). ***On the linearity of large non-linear models : when and why the tangent kernel is constant***.

NTK limit :

If network wide enough then $\Theta[u_{\theta_k}]$ and $\Theta[\mathcal{D}u_{\theta_k}]$ approximately constant for linear $\mathcal{D}$

Wang, Yu, Perdikaris (2022). ***When and why PINNs fail to train : A neural tangent kernel perspective***.

Choosing λ in $L(\theta) = R(\theta) + \lambda B(\theta)$

Ideally : choose λ to attain $\min_{\lambda} \kappa(A(\lambda))$

Existing approaches :

- Choose λ based on $\nabla_\theta R(\theta)$ and $\nabla_\theta B(\theta)$

  Wang, Teng, Perdikaris (2021). ***Understanding and Mitigating Gradient Flow Pathologies in PINNs.***
- Choose λ based on NTK PINN matrix

  Wang, Yu, Perdikaris (2022). ***When and why PINNs fail to train : A neural tangent kernel perspective.***

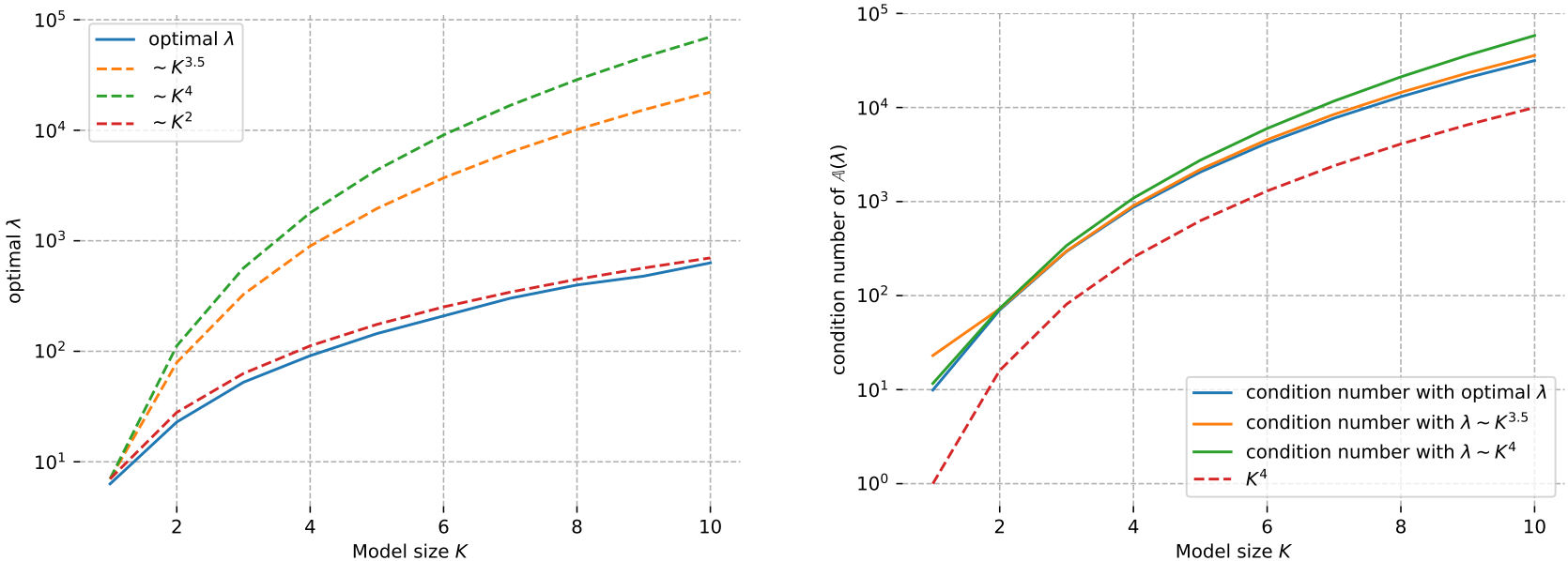


Figure 5 – *Left :* optimal λ vs. K. *Right :* $\kappa(A(\lambda))$ vs. K and λ.

Choosing λ in $L(\theta) = R(\theta) + \lambda B(\theta)$

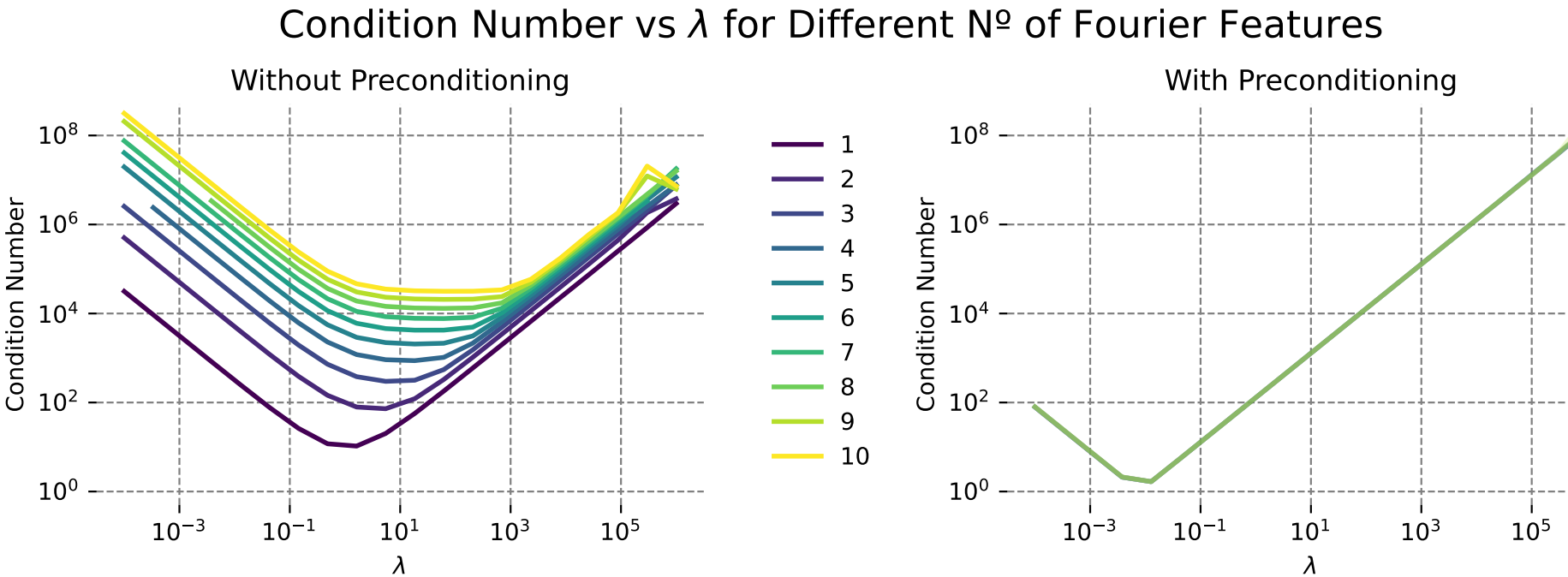


Figure 6 – Sensitivity of condition number on λ for different model sizes K.

Preconditioning linear models

Example (Poisson + Dirichlet BC) : 1D with $\mathcal{D} = \dfrac{d^2}{dx^2}$ on $(-\pi, \pi)$

Model : $u_\theta(x) = \displaystyle\sum_{k=-K}^{K} \theta_k \phi_k(x)$, with for $1 \leq k \leq K$,

$$\phi_0(x) = \tfrac{1}{\sqrt{2\pi}}, \quad \phi_{-k}(x) = \tfrac{1}{\sqrt{\pi}} \cos(kx), \quad \phi_k(x) = \tfrac{1}{\sqrt{\pi}} \sin(kx)$$

Resulting matrix :

$$A(\lambda) = \mathbb{D} + \lambda v v^\top$$

with $\mathbb{D}$ diagonal with $\mathbb{D}_{kk} = k^4$ and $v = \phi(\pi)$

Preconditioning : set $\mathbb{P}_{kk} = 1/k^2$ for $k \neq 0$ and $\mathbb{P}_{00} = \gamma \in \mathbb{R}$

$$\tilde{A}(\lambda, \gamma) = \mathbb{P}\mathbb{D}\mathbb{P}^\top + \lambda \mathbb{P} v (\mathbb{P} v)^\top$$

Preconditioning linear models

Preconditioned linear model : $\gamma\theta_0\phi_0(x) + \sum_{k\neq 0} \frac{\theta_k}{k^2}\phi_k(x)$

Theorem

The following statements hold for every model size $K \in \mathbb{N}$:

1. Condition number of unpreconditioned matrix satisfies

$$\kappa(A(\lambda)) \geq K^4$$

2. There is a constant $C(\lambda, \gamma) > 0$ (independent of K) with $\kappa(\tilde{A}(\lambda, \gamma) \leq C$.
3. It holds that $\kappa(\tilde{A}(2\pi/\gamma^2, \gamma)) = 1 + O(1/\gamma)$ and hence

$$\lim_{\gamma \to +\infty} \kappa(\tilde{A}(2\pi/\gamma^2, \gamma)) = 1.$$

Linear advection equation with neural network

Gradient preconditioning : $\hat{\theta}_{k+1} = \hat{\theta}_k - \eta(A + \epsilon \mathrm{Id})^{-1} \nabla_\theta L(\hat{\theta}_k)$

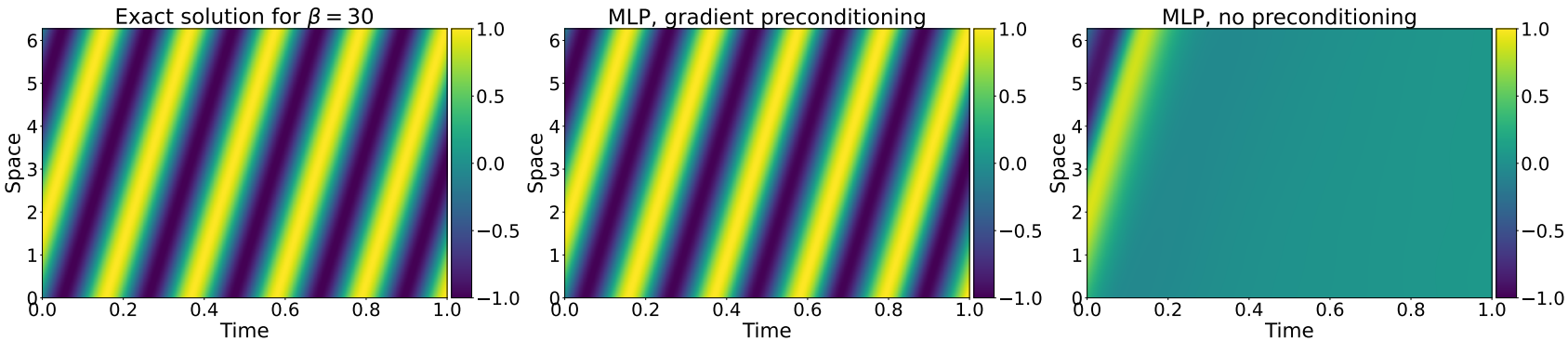


Figure 7 – Comparison of the solutions for the shallow MLP in the advection problem using preconditioned SGD and Adam with $\beta = 30$.

Preconditioning neural networks

Theory suggest two approaches :

1. Change NN architecture / initialization / loss / ...
   - Let u be NN, Φ FF and $\hat{\Phi}$ preconditioned FF
   - Idea : replace $u \circ \Phi$ with $u \circ \hat{\Phi}$
   - Change initialization
   - ...

2. Precondition gradient in GD

$$\hat{\theta}_{k+1} = \hat{\theta}_k - \eta \mathbb{P}\mathbb{P}^\top \nabla_\theta L(\hat{\theta}_k) = \hat{\theta}_k - \eta A^{-1} \nabla_\theta L(\hat{\theta}_k)$$

   - e.g. energy natural gradient descent
   - difficult in practice

Preconditioning to improve training

General idea : choose model u_θ (and hence $\mathscr{T}$) such that

$$\mathscr{T} \approx \mathcal{A}^{-1} \quad \text{and hence} \quad \kappa(\mathcal{A} \circ \mathscr{T}) \approx 1$$

If $\mathcal{A}$ has eigenvectors ψ_k and eigenvalues ω_k then ideally $\mathscr{T}\psi_k = \frac{1}{\omega_k}\psi_k$

Two equivalent ways to do this

- Set $\hat{\theta} := \mathbb{P}\theta \quad \Rightarrow \quad \hat{T}\,\hat{T}^*$ for which $\kappa(\mathcal{A} \circ \hat{T}\,\hat{T}^*) = \kappa(\hat{A})$,

$$\hat{A} = \mathbb{P}^\top A \mathbb{P}$$

- Preconditioning the gradient :

$$\hat{\theta}_{k+1} := \mathbb{P}\theta_{k+1} = \mathbb{P}\theta_k - \eta\mathbb{P}\mathbb{P}^\top \nabla_\theta L(\mathbb{P}\theta_k) = \hat{\theta}_k - \eta\mathbb{P}\mathbb{P}^\top \nabla_\theta L(\hat{\theta}_k)$$