Neural representation for PDEs

A. Belières[2], M. Duprez[1], E. Franck[12], F. Lecourtier[1],
V. Lleras[3], V. Michel-Dansac[12], Y. Privat[4]

28 Mars 2024
Journée thématiques MIRES, Réseaux de neurones et applications

[1]Inria Nancy Grand Est, France
[2]IRMA, Strasbourg university, France
[3]Montpellier University, France
[3]Mines-Nancy, France

Outline

Numerical Methods and implicit neural representation

Parametric models

- We consider a unknown function

$$y = f(x)$$

with $x \in V \subset \mathbb{R}^d$ and $y \in W \subset \mathbb{R}^p$.

Objective

- find $f_h \in H$ an approximation of f with H a functional space.
- **Difficulty**: it is a infinite dimensional problem.

Solutions parametric models

- We consider a function f_θ composed of known elementary functions and n unknown parameters θ_i
- The problem becomes : find $f_\theta \in H_n$ an approximation of f with H_n a finite dimensional functional space.
- It is equivalent to

$$\text{Find } \theta, \text{ such that } \| f_\theta - f \|_H \leq \epsilon$$

- **Main Question**: How determinate θ ?
- Example in the following. We want approximate the temperature in a Room:

$$T(t,x), \quad x \in \Omega \in \mathbb{R}^3, t \in \mathbb{R}^+$$

ML and regression

ML regression approach

- We have data and we use it to construct the parametric model which approach our function T

- We assume that we known: $\{(x_1, t_1, T_1), ...(x_N, t_N, T_N)\}$ such that

$$T_i = T(t_i, x_i) + \epsilon_i$$

with ϵ_i a noise.

- To approximate the temperature function we propose to approximate correctly our data examples.
- It is equivalent to solve:

$$\min_\theta \sum_{i=1}^{N} d(T_i, f_\theta(t_i, x_i))$$

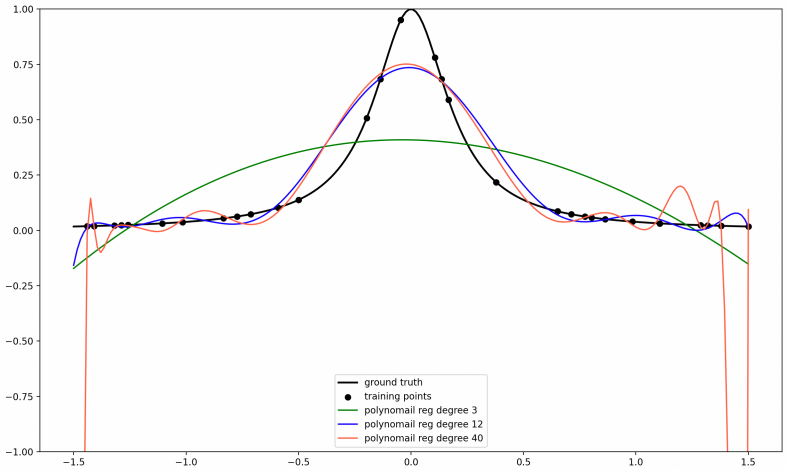
with d a distance like euclidian norm.

Questions in ML

- Which parametric model ?
- Generalization for input outside of the data set (overfitting) ?
- Robustness to the noise ?
- How collect, process the date ?

Models and garanties

- We consider: $y = f(\mathbf{x})$ with $\mathbf{x} = (x^1, ..., x^d) \in \mathbb{R}^d$
- Models:
  - Linear model:
$$\sum_{i=1}^{d} \theta_i x^i$$
  - Polynomial model:
$$\sum_{i=1}^{n} \theta_i P_i(\mathbf{x})$$
  - Kernel model:
$$\sum_{i=1}^{N} \theta_i K(\mathbf{x}, \mathbf{x}_i)$$
    with $\mathbf{x}_i$ a data and K a symmetric kernel.



- Polynomial regression of the Runge function

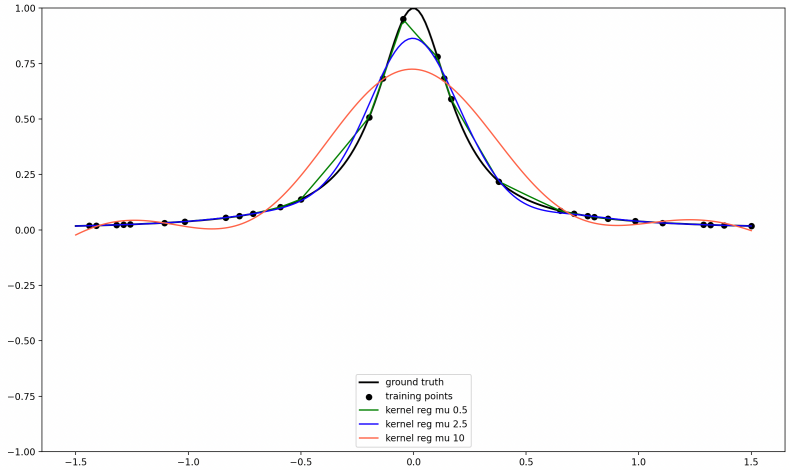
- **Garanties**: For $d = \| x - y \|_2^2$ the minimization problem is convex and admit a unique solution if you have sufficient number of data.
- For nonlinear models compared to the inputs more you have data and parameters more you will accurate.

Curse of dimensionality

The number of data needed to approximate well the function grows up exponentially with the dimension d

Models and garanties

- We consider: $y = f(\mathbf{x})$ with $\mathbf{x} = (x^1, ..., x^d) \in \mathbb{R}^d$
- Models:
  - Linear model:
$$\sum_{i=1}^{d} \theta_i x^i$$
  - Polynomial model:
$$\sum_{i=1}^{n} \theta_i P_i(\mathbf{x})$$
  - Kernel model:
$$\sum_{i=1}^{N} \theta_i K(\mathbf{x}, \mathbf{x}_i)$$
  with $\mathbf{x}_i$ a data and K a symmetric kernel.



- Kernel regression of the Runge function

- **Garanties**: For $d = \| x - y \|_2^2$ the minimization problem is convex and admit a unique solution if you have sufficient number of data.
- For nonlinear models compared to the inputs more you have data and parameters more you will accurate.

Curse of dimensionality

The number of data needed to approximate well the function grows up exponentially with the dimension d

Numerical methods

Principe of numerical methods

- **Same objective than ML**: construct a parametric model approaching T.
- no data but a strong constrain on the function: the equation

- Equation for temperature evolution:

$$\begin{cases} L_{t,x}u = \partial_t T - \Delta T = f(x) \\ T(t=0,x) = T_0(x) \\ T(x) = g \text{ on } \partial\Omega \end{cases}$$

- **Numerical method**: choose a parametric model, transform the equation/constrain on the function on a equation/constrain on the parameters.

Important: convergence

For numerical methods, we want that $\| f_\theta - f \|_h \to 0$, when , $n \to \infty$ with n the number of parameters (call degrees of freedom).

- For the three next slides, i consider only a spatial problem like $-\Delta T = f(x)$

Parametric models

- In all the classical numerical method we choose: $f_\theta = \sum_{i=1}^{n} \theta_i \phi_i(x)$
- How construct ϕ_i ?

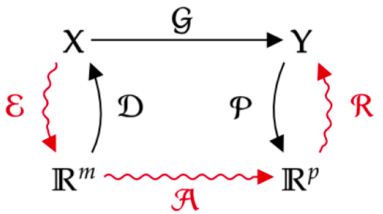
Mesh based methods

Polynomial Lagrange interpolation

We consider a domain $[a, b]$. There exists a polynomial P of degree k such that, for any $f \in C^0([a, b])$,

$$|f(x) - P(x)| \leq |b - a|^k \max_{x \in [a,b]} |f^{k+1}(x)|.$$

- On small domains ($|b - a| \ll 1$) or for large k, this polynomial gives a very good approximation of any continuous function.
- Very high degrees k can generate oscillations (like in ML).
- To obtain good approximation: we introduce a mesh and a cell-wise polynomial approximation



- Possible since contrary to ML, the domain of inputs is always well-known.

First step: choose a parametric function

We define a mesh by splitting the geometry in small sub-intervals $[x_k, x_{k+1}]$, and we propose the following candidate to approximate the PDE solution T

$$T_{|[x_k,x_{k+1}]}(t, x) = \sum_{j=1}^{k} \theta_k^j \phi_j(x).$$

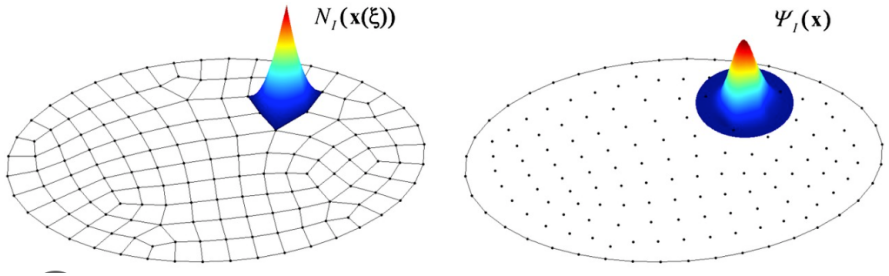
This is a piecewise polynomial representation.

Classical numerical methods, encoder and decoder

Parametric model for all numerical methods;

$$f_\theta = \sum_{i=1}^{n} \theta_i \phi_i(x)$$

- Classical mesh based methods:
  - **Finite element**: C^p continuity between the cells (depend of the finite element) so $\phi_i(x)$ piecewise polynomial.
  - **DG**: discontinuity between the cell so $\phi_i(x) = p_j(x)\chi_{x\in\Omega_i}$.
  - **DG Treffz**: same as DG but non-polynomial.
  - **Finite difference**: punctual value so $\phi_i(x) = \delta_{x_i}(x)$ with x_i a mesh node.
- Classical mesh free methods:
  - **Spectral**: we use Hilbert basis so $\phi_i(x) = sin(2\pi k_i x)$ for example (same with Hermite, Laguerre, Legendre polynomiales).
  - **Radial basis**: we use radial basis so for example $\phi_i = \phi(\mid x - x_i)$ with ϕ a Gaussian or $\frac{1}{1+\sigma^2 x^2}$.



How determinate the degree of freedom

General method

The aim is to transform the PDE on T into a equation on θ (DOF).

- We note $V_\theta = Span\{f_\theta, \text{such that }, \theta \in V \in \mathbb{R}^n\}$
- First approach: Galerkin
  - Rewrite the problem:

$$-\Delta T(x) = f(x), \Longleftrightarrow \min_{T \in H} \int_\Omega \left(\mid \nabla T(x) \mid^2 - f(x) T(x) \right) dx$$

  - Galerkin projection:

$$\min_{T_\theta \in V_\theta} \int_\Omega \left(\mid \nabla T_\theta(x) \mid^2 - f(x) T_\theta(x) \right) dx$$

- The problem is quadratic in θ. The parameters which put the gradient at zero satisfy

$$\int_\Omega (-\Delta T_\theta(x) - f)\phi_i(x) = 0, \quad \forall i \in \{1, ..., n\}$$

- Since we can compute exactly the derivative and numerically the integral we precompute everything (after in general a integration by part) to obtain

$$A\theta = b$$

- Second approach: Least square Galerkin projection

$$\min_{\theta \in V} \int_\Omega \mid -\Delta T_\theta - f \mid^2 dx$$

Time case

Space time methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i \phi_i(t,x)$$

- The time equation have no equivalent minimization form so we use the Least square Galerkin projection.
- In practice we compute the gradient and obtain a large system to invert since n is large

Space methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i(t) \phi_i(x)$$

- We consider Least square Galerkin:

$$\min_{\theta(t)\in V} \int_\Omega \mid \partial_t T_{\theta(t)(x)} - \Delta T_{\theta(t)(x)} - f(t,x) \mid^2 dx$$

Time case

Space time methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i \phi_i(t,x)$$

- The time equation have no equivalent minimization form so we use the Least square Galerkin projection.
- In practice we compute the gradient and obtain a large system to invert since n is large

Space methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i(t) \phi_i(x)$$

- We discretize on time (here a Euler method):

$$\min_{\theta(t^{n+1}) \in V} \int_\Omega \mid T_{\theta(t_{n+1})(x)} - T_{\theta(t_n)(x)} - \Delta t(\Delta T_{\theta(t)(x)} - f(t,x)) \mid^2 dx$$

Time case

Space time methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i \phi_i(t,x)$$

- The time equation have no equivalent minimization form so we use the Least square Galerkin projection.
- In practice we compute the gradient and obtain a large system to invert since n is large

Space methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i(t) \phi_i(x)$$

- it gives a succession of Galerkin (L^2) projection on the spatial approximation space:

$$\min_{\theta(t^{n+1}) \in V} \int_\Omega | \left(\Phi(x), \theta(t_{n+1})\right) - \left(\Phi(x), \theta(t_n)\right) - \Delta t (\Delta T_{\theta(t)(x)} - f(t,x)) |^2 \, dx$$

- This projection are smaller an faster than in the space time methods.
-

Time case

Space time methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i \phi_i(t,x)$$

- The time equation have no equivalent minimization form so we use the Least square Galerkin projection.
- In practice we compute the gradient and obtain a large system to invert since n is large

Space methods

We use the parametric model:

$$f_\theta = \sum_{i=1}^{n} \theta_i(t) \phi_i(x)$$

- Computing the gradient and $\nabla_\theta J(\theta) = 0$:

$$\left(\int_\Omega \Phi \otimes \Phi\right) \theta(t_{n+1}) = \left(\int_\Omega \Phi \otimes \Phi\right) \theta(t_n) - \Delta t \left(\int_\Omega \Phi(x)(\Delta T_{\theta(t)(x)} - f(t,x))\right)$$

- it is the equivalent to the normal equation in infinite dimension for Least square problem.

Garanties

Essential point

The space V_θ is a a vectorial space. So the projector is on subspace is unique (projection on convexe subspace of Hilbert theorem). It allows to assure that the problem on parameters admit also a unique solution.

Convergence

The previous property coupled the approximation theorem of polynomial or Hilbert basis allows to assure that

$$\| f_\theta - f \|_h \to 0, \text{when } , n \to \infty$$

Curse of dimensionality

For mesh based approaches

$$\| f_\theta - f \|_H \leq Ch^p$$

with h characteristic size of the cells and the number of cell $N = O(\frac{1}{h^d})$. For that we need p polynomial by cell and direction so $O(p^d)$ parameters by cell. There is also similar problem for mesh less methods.

Neural representation in ML and numerics

Deep ML, nonlinear model and manifold

Key point

All the parametric models introduced for ML or numerical methods are linear compared to the parameters and gives finite dimension function vectorial space

Deep learning

The rupture associated to the deep learning is to use massively nonlinear compared to the parameters which gives finite dimension function manifold

Projection on manifold

How project on manifold ? Not uniqueness ? The convex optimisation problem are replaced by non-convex problem. So there is less guaranties on the results.

Nonlinear models

- Nonlinear version of classical models: f is represented by the DoF α_i, μ_i, ω_i or Σ_i:

$$f(x;\alpha,\mu,\Sigma)=\sum_{i=1}\alpha_i e^{(x-\mu_i)\Sigma_i^{-1}(x-\mu_i)},\quad f(x;\alpha,\omega)=\sum_{i=1}\alpha_i sin(\omega_i x)$$

- Neural networks (NN).

Layer

A layer is a function $L_l(\mathbf{x}_l):\mathbb{R}^{d_l}\rightarrow\mathbb{R}^{d_{l+1}}$ given by

$$L_l(\mathbf{x}_l)=\sigma(A_l\mathbf{x}_l+\mathbf{b}_l),$$

$A_l\in\mathbb{R}^{d_{l+1},d_l}$, $\mathbf{b}\in\mathbb{R}^{d_{l+1}}$ and $\sigma()$ a nonlinear function applied component by component.

Neural network

A neural network is parametric function obtained by composition of layers:

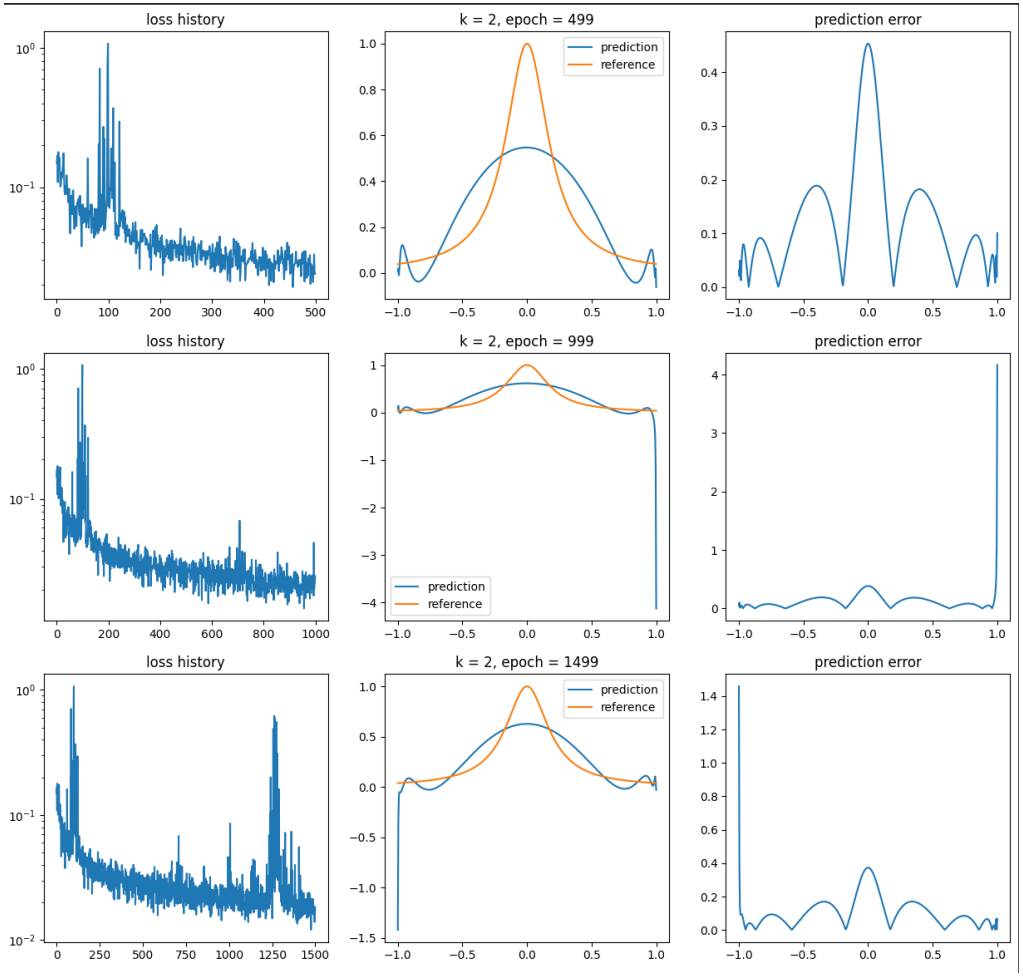
$$f_\theta(\mathbf{x})=L_n\circ\ldots.\circ L_1(\mathbf{x})$$

with θ the trainable parameters composed of all the matrices $A_{l,l+1}$ and biases $\mathbf{b}_l$.

- **Go to nonlinear models**: would allows to use less parameters and data.
- **Go to nonlinear models** allows to use NN which are: accurate global model, low frequency (better for generalization) and able to deal with large dimension.

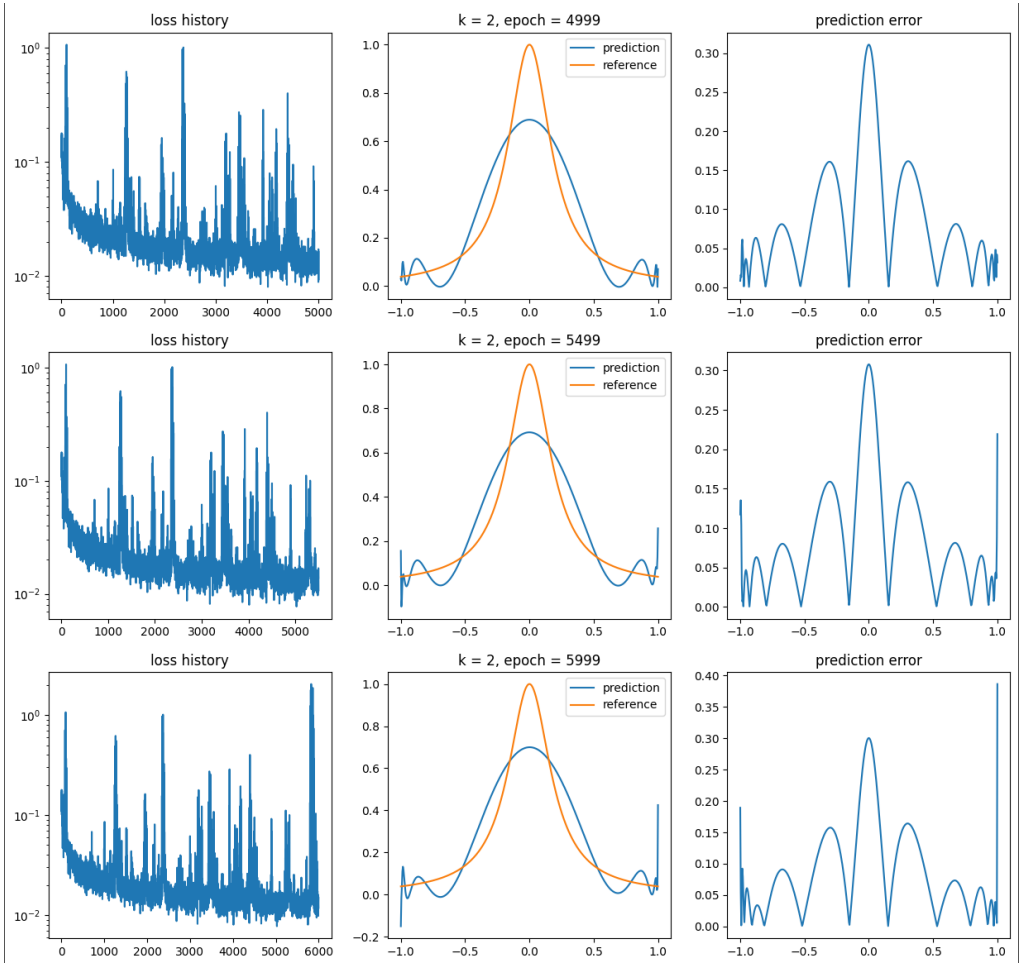
NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



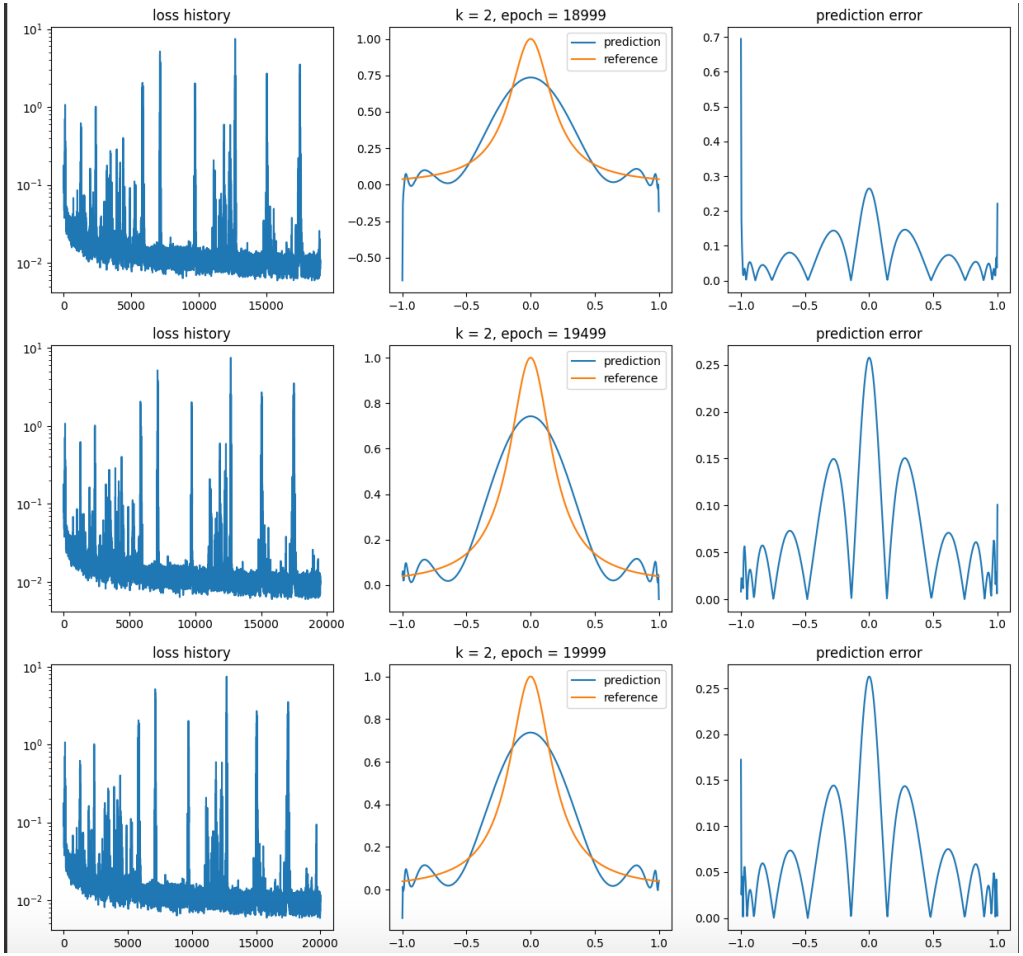
NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



NN vs Polynomial

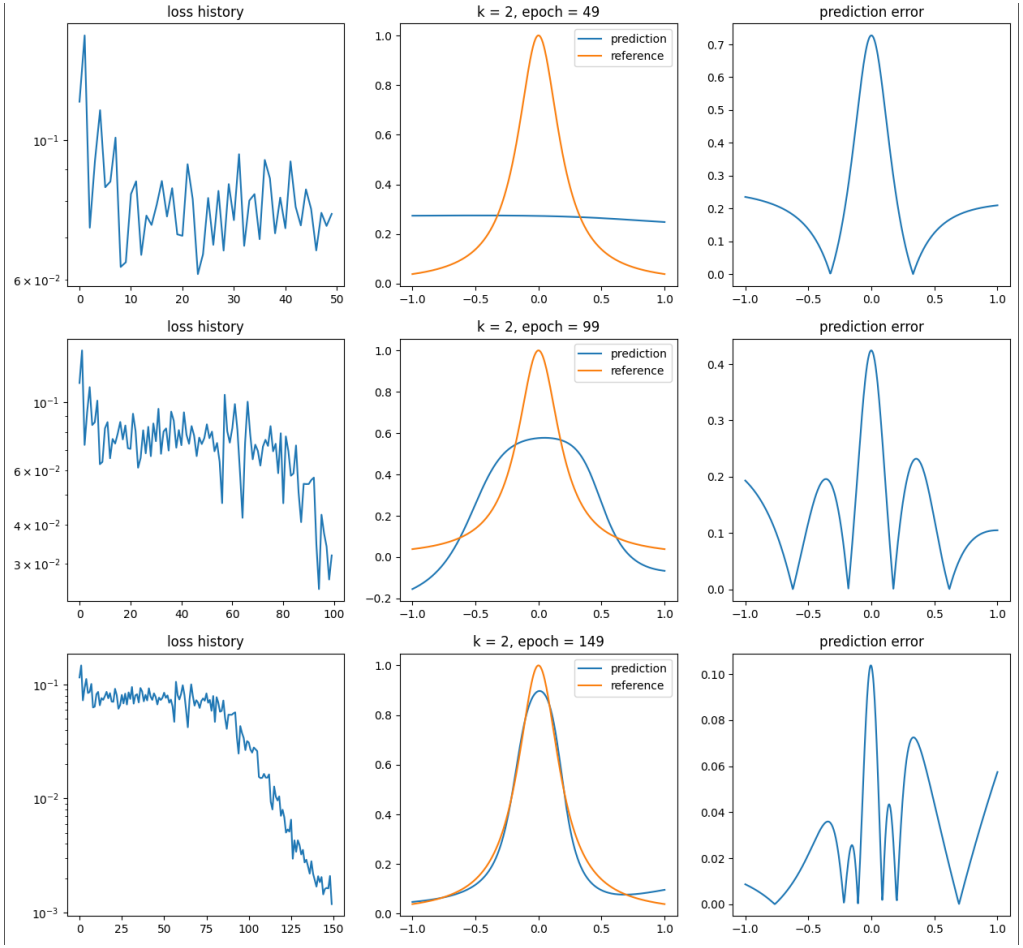
- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



- The polynomial model tends to oscillate in the over parameterized regime. Problematic for overfitting.

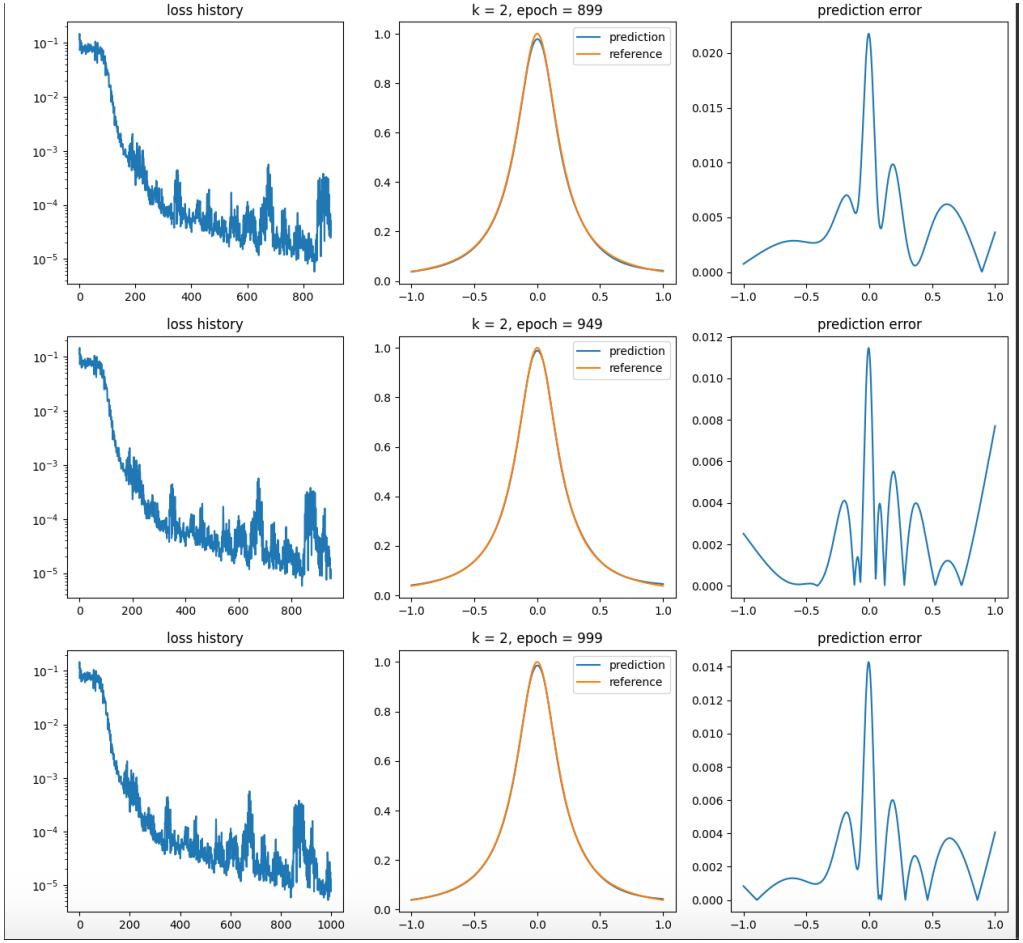
NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



- The ANN generates very smooth/low frequency approximations.
- It is related to the spectral bias. The low frequencies are learned before the high frequencies.
- Seems very helpful to use it for global and high dimensional representation.

Space-time approach: PINNs I

Neural methods

The PINNs and Neural Galerkin approaches use exactly the same strategy than classical numerical methods but project on manifold associated to nonlinear parametric models compared to the parameters

Idea of PINNs

- For u in some function space $\mathcal{H}$, we wish to solve the following PDE:

$$\partial_t u = \mathcal{F}(u, \nabla u, \Delta u) = F(u).$$

- Classical representation for space-time approach: $u(t,x) = \sum_{i=1}^{N} \theta_i \phi_i(x,t)$
- Deep representation: $u(t,x) = u_{nn}(x,t;\theta)$ with u_{nn} a NN with trainable parameters θ.

Which projection

- Galerkin projection is just valid for elliptic equations with energetic form.
- More general: Least square Galerkin. We minimize the least square residue of the restricted to the manifold associated by our chosen neural architecture.

Space-time approach: PINNs II

- We define the residual of the PDE:

$$R(t,x) = \partial_t u_{nn}(t,x;\theta) - \mathcal{F}(u_{nn}(t,x;\theta), \partial_x u_{nn}(t,x;\theta), \partial_{xx} u_{nn}(t,x;\theta))$$

- To learn the parameters θ in $u_{nn}(t,x;\theta)$, we minimize:

$$\theta = \arg\min_{\theta} \Big(J_r(\theta) + J_b(\theta) + J_i(\theta) \Big),$$

with

$$J_r(\theta) = \int_0^T \int_\Omega |R(t,x)|^2 dx dt$$

and

$$J_b(\theta) = \int_0^T \int_{\partial\Omega} \|u_{nn}(t,x;\theta) - g(x)\|_2^2 dx dt, \quad J_i(\theta) = \int_\Omega \|u_{nn}(0,x;\theta) - u_0(x)\|_2^2 dx.$$

- If these residuals are all equal to zero, then $u_{nn}(t,x;\theta)$ is a solution of the PDE.
- To complete the determination of the method, we need a way to compute the integrals. In practice we use Monte Carlo.
- **Important point**: the derivatives are computed exactly using automatic differentiation tools and back propagation. Valid for any decoder proposed.

Space-time approach: PINNs II

- We define the residual of the PDE:

$$R(t,x) = \partial_t u_{nn}(t,x;\theta) - \mathcal{F}(u_{nn}(t,x;\theta), \partial_x u_{nn}(t,x;\theta), \partial_{xx} u_{nn}(t,x;\theta))$$

- To learn the parameters θ in $u_{nn}(t,x;\theta)$, we minimize:

$$\theta = \arg\min_{\theta} \Big(J_r(\theta) + J_b(\theta) + J_i(\theta) \Big),$$

with

$$J_r(\theta) = \sum_{n=1}^{N} \sum_{i=1}^{N} |R(t_n, x_i)|^2$$

with (t_n, x_i) sampled uniformly or through importance sampling, and

$$J_b(\theta) = \sum_{n=1}^{N_b} \sum_{i=1}^{N_b} |u_{nn}(t_n, x_i; \boldsymbol{\theta}) - g(x_i)|^2, \quad J_i(\theta) = \sum_{i=1}^{N_i} |u_{nn}(0, x_i; \boldsymbol{\theta}) - u_0(x_i)|^2.$$

- If these residuals are all equal to zero, then $u_{nn}(t,x;\theta)$ is a solution of the PDE.
- To complete the determination of the method, we need a way to compute the integrals. In practice we use Monte Carlo.
- **Important point**: the derivatives are computed exactly using automatic differentiation tools and back propagation. Valid for any decoder proposed.

PINNs for parametric PDEs

- **Advantages of PINNs**: mesh-less approach, not too sensitive to the dimension.
- **Drawbacks of PINNs**: they are often not competitive with classical methods.
- Interesting possibility: use the strengths of PINNs to solve PDEs parameterized by some $\boldsymbol{\mu}$.
- The neural network becomes $u_{nn}(t, x, \boldsymbol{\mu}; \theta)$.

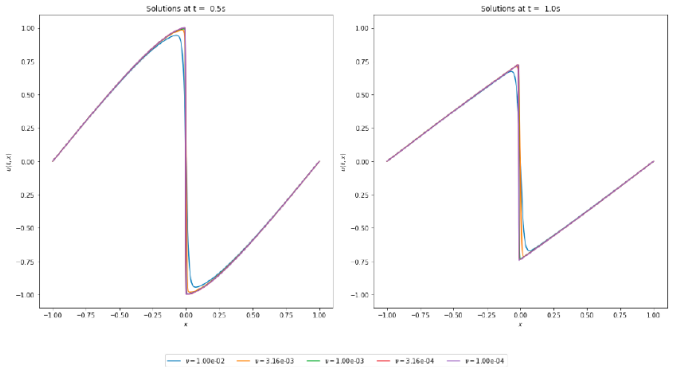
New Optimization problem for PINNs

$$\min_{\theta} J_r(\theta) + \ldots, , \quad \text{with}$$

$$J_r(\theta) = \int_{V_\mu} \int_0^T \int_\Omega \left\| \partial_t u_{nn} - \mathcal{L}\left(u_{nn}(t, x, \boldsymbol{\mu}), \partial_x u_{nn}(t, x, \boldsymbol{\mu}), \partial_{xx} u_{nn}(t, x, \boldsymbol{\mu})\right) \right\|_2^2 dxdt$$

with V_μ a subspace of the parameters $\boldsymbol{\mu}$.

- Application to the Burgers equations with many viscosities $[10^{-2}, 10^{-4}]$:



- Training for $\mu = 10^{-4}$: 2h. Training for the full viscosity subset: 2h.

Spatial approach: Neural Galerkin I

- We solve the following PDE:

$$\partial_t u = \mathcal{F}(u, \nabla u, \Delta u) = F(u).$$

- Classical representation: $u(t,x) = \sum_{i=1}^{N} \theta_i(t)\phi_i(x)$
- Deep representation: $u(t,x) = u_{nn}(x;\theta(t))$ with u_{nn} a neural network, with parameters $\theta(t)$, taking x as input.
- We use the same strategy as before: we begin with Lest square Galerkin Projection

$$\min_{\theta(t)\in\mathbb{R}^n} \int_\Omega \mid \partial_t u_{nn}(x;\theta(t)) - F(u_{nn}(x;\theta(t))) \mid^2 dx$$

- We discretize in time

$$\min_{\theta(t_{n+1})\in\mathbb{R}^n} \int_\Omega \mid u_{nn}(x;\theta(t_{n+1})) - u_{nn}(x;\theta(t_n)) - \Delta t F(u_{nn}(x;\theta(t))) \mid^2 dx$$

- Here we solve a succession of nonlinear optimization problems (similar to linear case). Since we initialize $\theta(t_{n+1})$ with $\theta(t_n)$ and the weights evolve slowly these optimization problems are fast to solve.
- We speak about "Discrete time PINN".

Spatial approach: Neural Galerkin II

- Variant: Neural galerkin. We linearize:

$$u_{nn}(x;\theta(t_{n+1})) \approx u_{nn}(x;\theta(t_n)) + (\theta_{t_{n+1}} - \theta_{t_n})\nabla_\theta u_{nn}(x;\theta(t_n))$$

  So we have

$$\min_{\theta(t_{n+1})\in\mathbb{R}^n} \int_\Omega \mid \nabla_\theta u_{nn}(x;\theta)\theta_{t_{n+1}} - \nabla_\theta u_{nn}(x;\theta)\theta_{t_n} - \Delta t F(u_{nn}(x;\theta(t))) \mid^2 dx$$

- Since the problem is quadratic in $\theta(t_{n+1})$ we can compute the solution which is given by

$$M(\theta(t_n))\theta_{t_{n+1}} = M(\theta(t_n))\theta_{t_n} - \Delta t f(\theta(t_n))$$

  with

$$M(\theta(t)) = \int_\Omega \nabla_\theta u_{nn}(x;\theta)\otimes\nabla_\theta u_{nn}(x;\theta)dx, \quad f(,\theta(t)) = \int_\Omega \nabla_\theta u_{nn}(x;\theta)F(u_{nn}(x;\theta))dx.$$

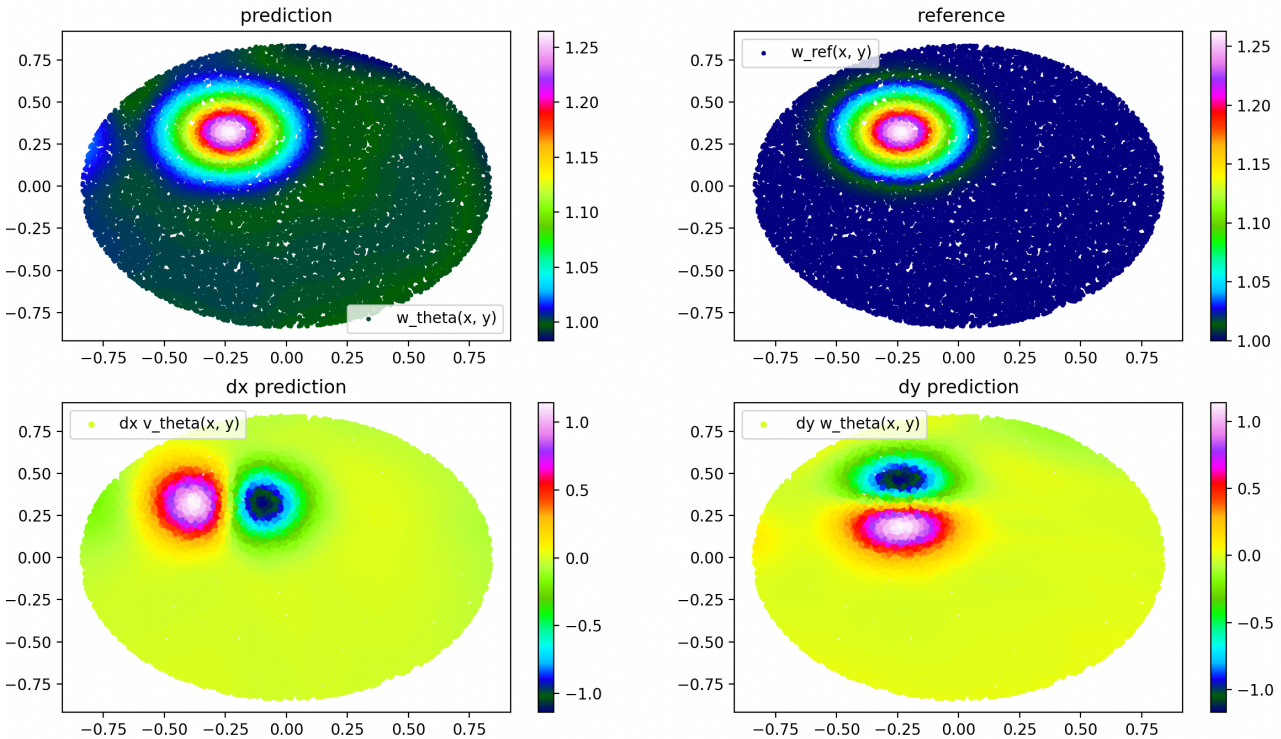
- How to estimate $M(\theta(t))$ and $F(x,\theta(t))$?
- **Firstly**: we need to differentiate the network with respect to θ and to x (in the function F). This can easily be done with automatic differentiation.
- **Secondly**: How to compute the integrals? Monte Carlo approach.
- So, we use (same for $f(\theta(t))$):

$$M(\theta(t)) \approx \sum_{i=1}^{N} \nabla_\theta u_{nn}(x_i;\theta) \otimes \nabla_\theta u_{nn}(x_i;\theta)$$

- Like in the case of PINNs, we can apply this framework to parametric PDEs and larger dimensions.

Spatial approach: Neural Galerkin III

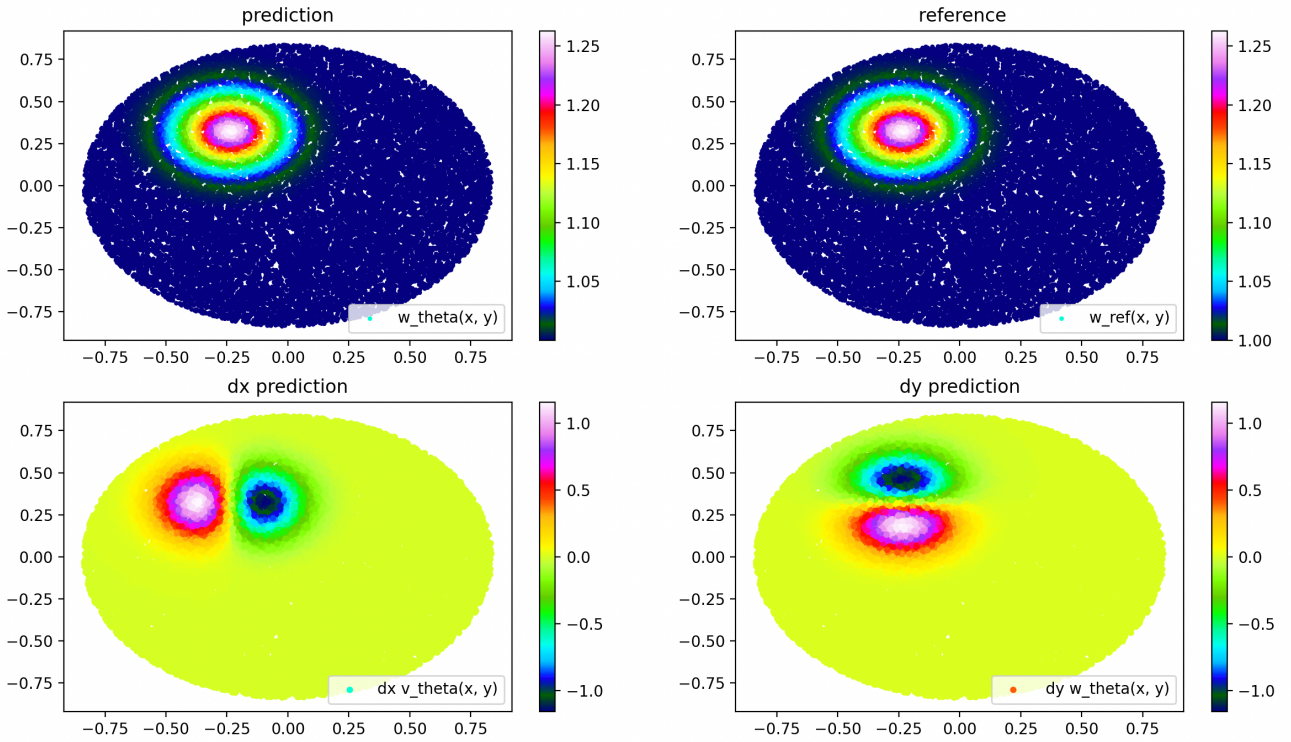
- We solve the advection-diffusion equation $\partial_t \rho + \boldsymbol{a} \cdot \nabla \rho = D \Delta \rho$ with a Gaussian function as initial condition.
- Case 1: with a neural network (2200 DOF)



- 5 minutes on CPU, MSE error around 0.0045.

Spatial approach: Neural Galerkin III

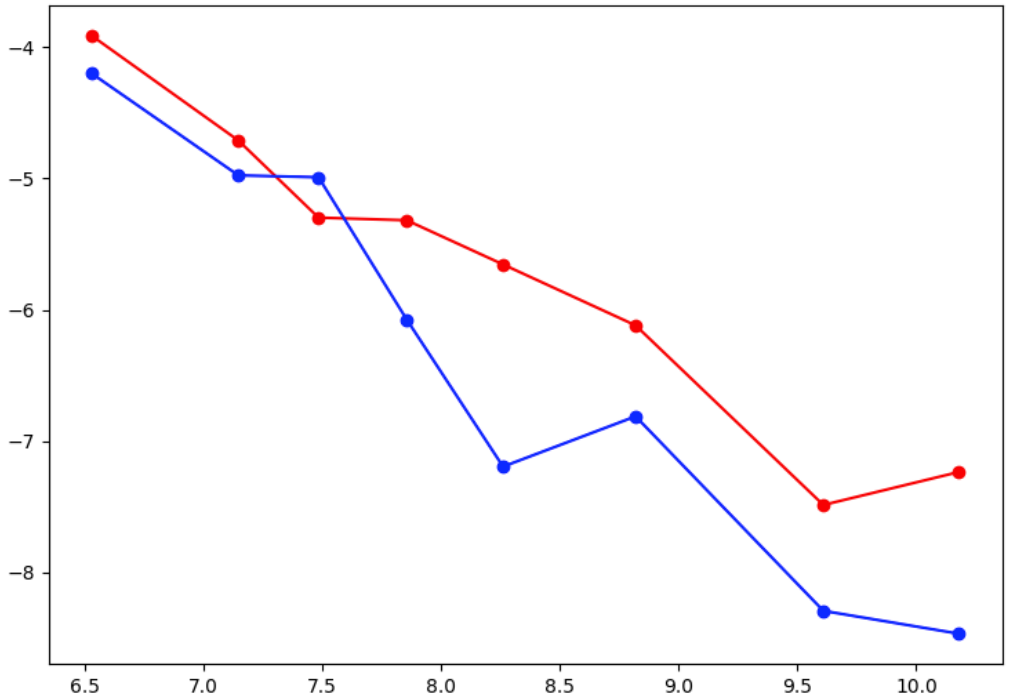
- We solve the advection-diffusion equation $\partial_t \rho + \boldsymbol{a} \cdot \nabla \rho = D \Delta \rho$ with a Gaussian function as initial condition.
- Case 2: with a Gaussian mixture (one Gaussian):



- 5 sec on CPU. MSE around 1.0^{-6}. Decoder perfect to represent this test case.

Convergence ?

- I solve a 2D laplacian with 5 layers neural network and increase the size (685 weights the smaller 26300 weights the larger).
- Two learning rates:



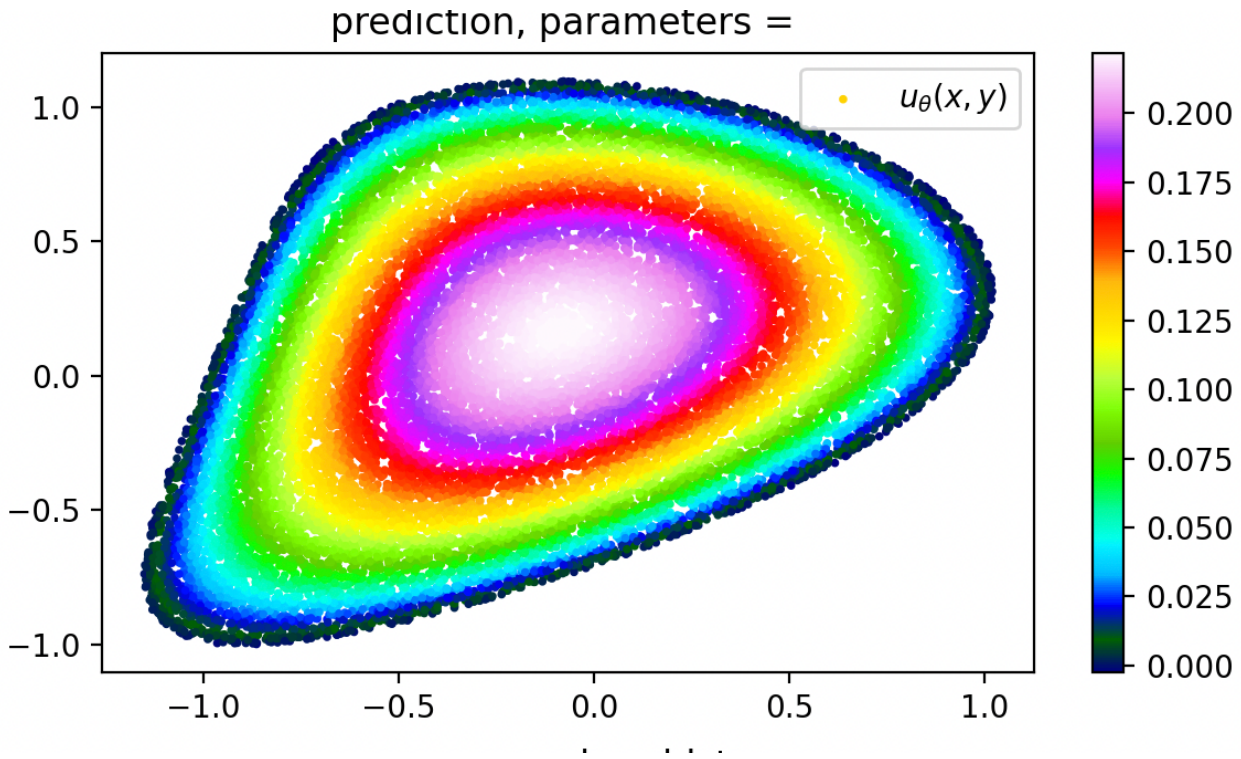
- The behavior of the error when we increase" the number of weights is complex".
- Sensitivity to the dimension. We use a network of 960 parameters for 1D/2D laplacian:

$$| u_{1D} |_{L^\infty} = 1.4e^{-4}, \quad | u_{2D} |_{L^\infty} = 3e^{-4}$$

How improve PINNs

How go complex geometry ? mapping

- **Claim on PINNs**: no mesh, so easy to go to complex geometries.
- **In practice**: No so easy. We need to find how sample in the geometry.
- First approach:
  - We are able to sample easily: quadrilateral, ellipse, cylinder etc
  - Using union/soustraction/intersection we can sample more complex domains.
- Second approach: mapping
  - We consider a simple domain Ω_0 and the target domain Ω
  - We assume that $\Omega = \phi(\Omega_0)$
  - We sample in Ω_0 and apply ϕ to the points sampled.



How go complex geometry ? level-set

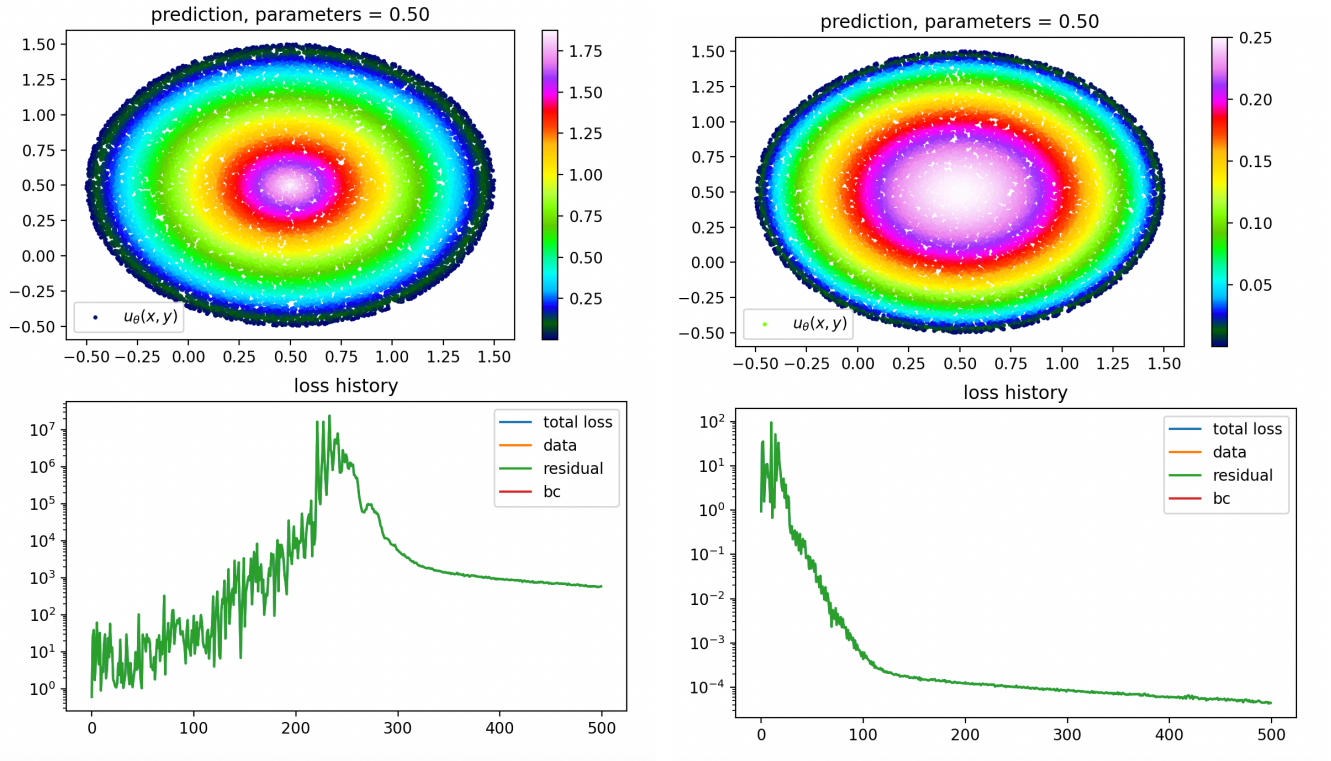
- We define the model by a level set $\phi(x)$ which satisfy

$$\phi(x) = 0, x \in \partial\Omega, \quad \phi(x) < 0, x \in \Omega, \quad \phi(x) > 0, x \in \mathbb{R}^n/\Omega,$$

- Sample is easy in this case. Allow to impose in hard the BC (example for Dirichlet):

$$u_\theta(x) = u_{nn,\theta}(x)\phi(x) + g(x)$$

- How construct ϕ? Classic level set: the signed distance function.



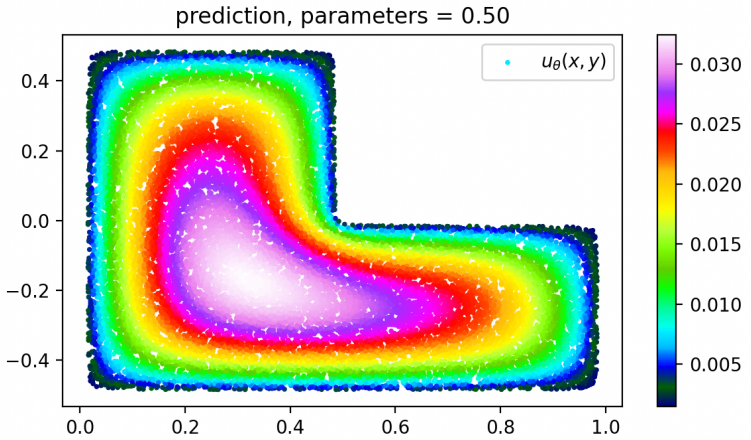
- Left: exact distance function, right: smooth levelset.

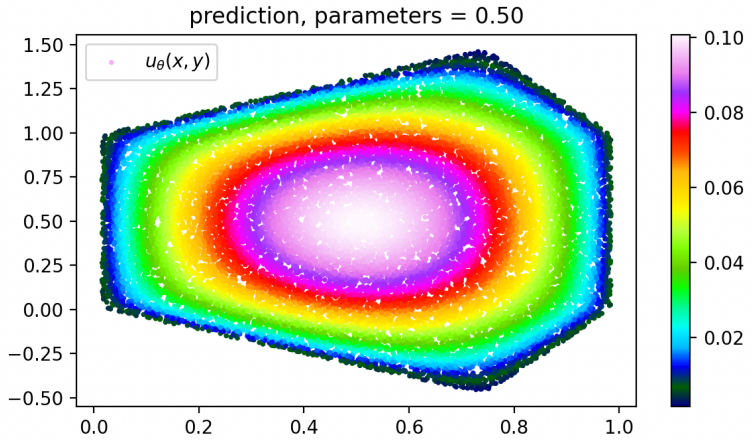
How go complex geometry ?

Remark on levelset

The exact distance function is a C^0 non C^1 function. The derivates explode.
If we impose the BC using the Distance function the network must compensate the singularity. For the BC we need regular level set.

- How construct smooth signed distance function ?
  - **First solution**: Approximation theory (*Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks*).
  - Δu can be singular at the boundary. Sampling at ϵ to the BC solve the problem.





- **Second solution**: learn the signed distance function. How make that ? with a PINNs.

How go complex geometry ?

Signed Distance function

If we have a boundary domain Γ, the SDF is solution to the Eikonal equation:

$$\begin{cases} |\nabla\phi(x)| = 1, & x \in [0,1]^d \\ \phi(x) = 0, & x \in \Gamma \\ (\nabla\phi(x), \mathbf{n}) = 0, & x \in \Gamma \end{cases}$$

- In practice we solve the Eikonal equation with PINNs
- To obtain a smooth SDF (important to impose strongly the BC) we add a penalization:

$$L_{penalize}(\theta) = \lambda \mid \partial_{xx}\phi(x) \mid^2$$

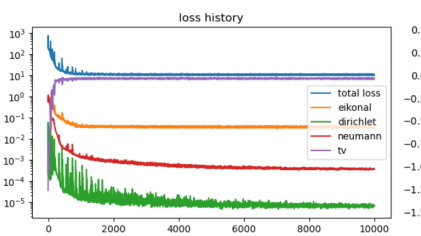


FIGURE 11 – Loss (tv=lap).

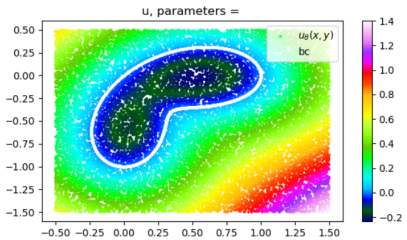


FIGURE 12 – Levelset ϕ_θ.

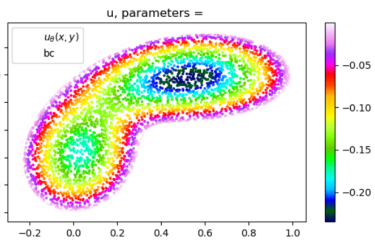
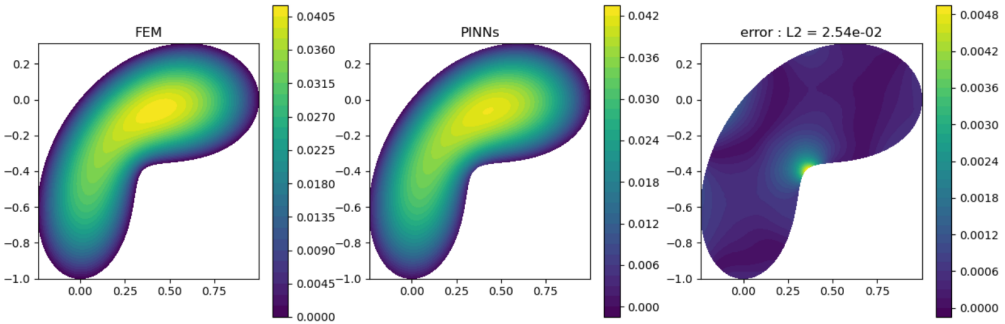


FIGURE 13 – Levelset $\phi_\theta < 0$.



How go complex geometry ?

Signed Distance function

If we have a boundary domain Γ, the SDF is solution to the Eikonal equation:

$$\begin{cases} |\nabla\phi(x)| = 1, & x \in [0,1]^d \\ \phi(x) = 0, & x \in \Gamma \\ (\nabla\phi(x), \mathbf{n}) = 0, & x \in \Gamma \end{cases}$$

- In practice we solve the Eikonal equation with PINNs
- To obtain a smooth SDF (important to impose strongly the BC) we add a penalization:

$$L_{penalize}(\theta) = \lambda \mid \partial_{xx}\phi(x) \mid^2$$

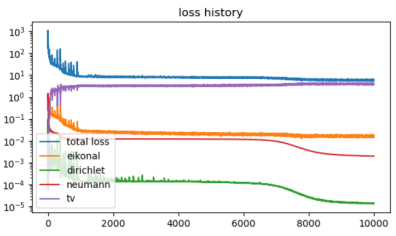


FIGURE 17 – Loss (tv=lap).

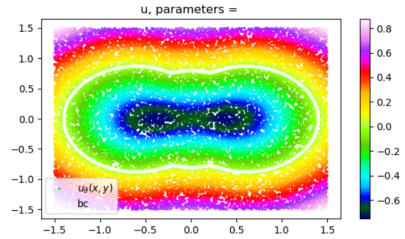


FIGURE 18 – Levelset ϕ_θ.

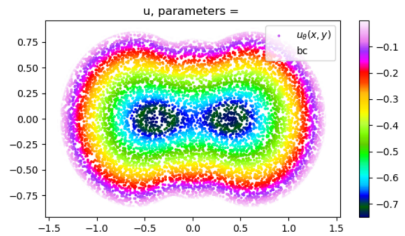
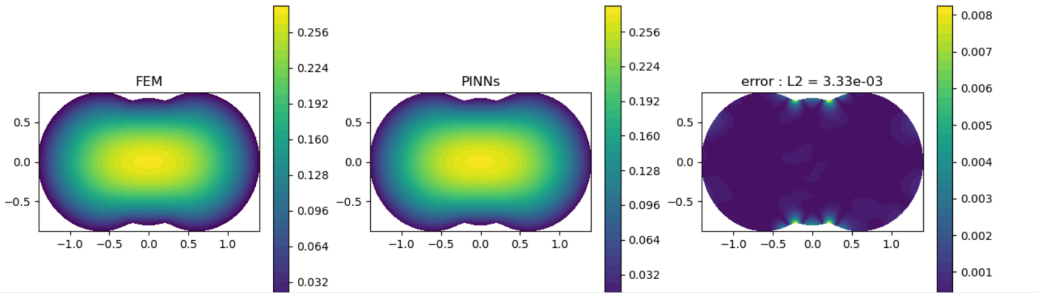
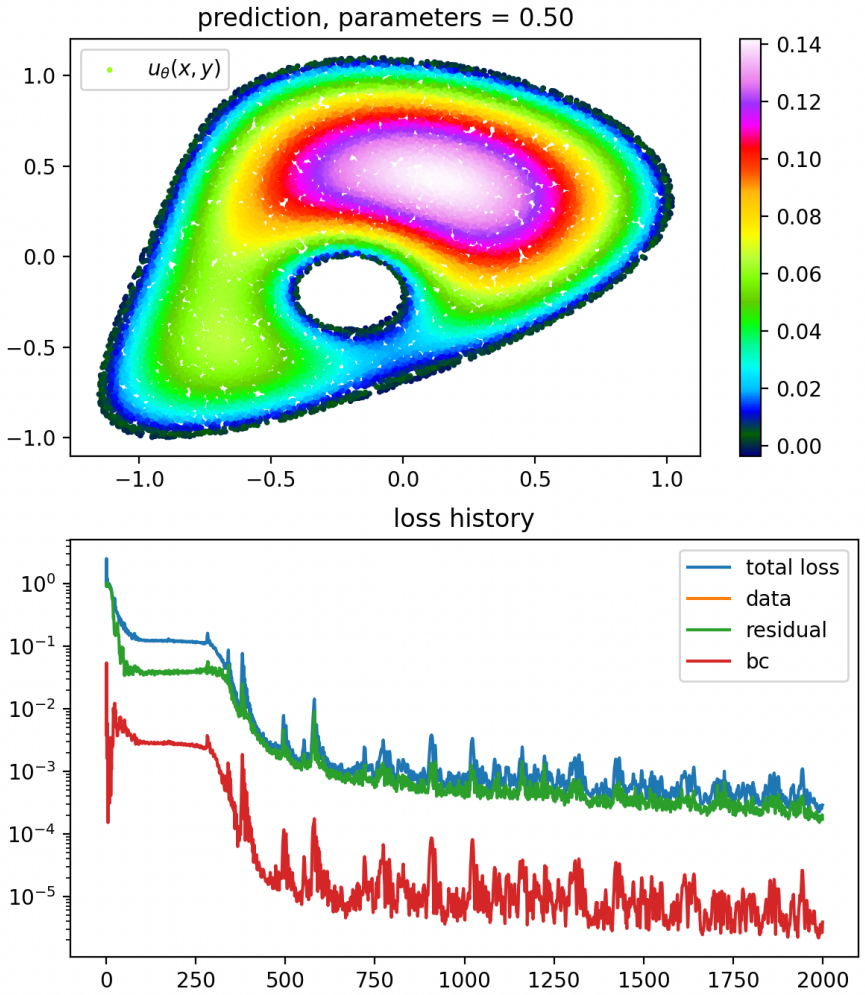


FIGURE 19 – Levelset $\phi_\theta < 0$.



How go complex geometry ?

- We can combine the options: mapping for the big domain. Level set for the holes.



NTK

- How study the learning dynamic and discover training bias: NTK theory.
- We call $\mathcal{L}(\theta)$ the loss and $f_\theta(x)$ the model. Continuous gradient descent:

$$\frac{d\theta(t)}{dt} = -\nabla_\theta \mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}(\nabla_\theta f_\theta(x_i))(\nabla_{f_\theta} l(f_\theta(x_i), y_i)$$

- We multiply by $\nabla_\theta f_\theta(x)$

$$\frac{df_{\theta(t)}(x)}{dt} = -\frac{1}{N}\sum_{i=1}^{N} K(x, x_i)(\nabla_{f_\theta} l(f_\theta(x_i), y_i)$$

with $K_{\theta(t)}(x, y) = (\nabla_\theta f_\theta(x))^T(\nabla_\theta f_\theta(y))$.

Theorem

In the limit n the number of neurons tends to infinity We have:

- $K_{\theta(0)}(x, y)$ deterministic at initialization, only determined by the model architecture
- $K_{\theta(t)}(x, y) = K_{\theta(0)}(x, y)$

- So we have:

$$f_{\theta(t)}(X) = (f_{\theta(0)}(X) - Y)e^{-\eta K_0(X,\bar{X})}$$

with X the evaluation points and $\bar{X}$ the training points, η the learning rates.

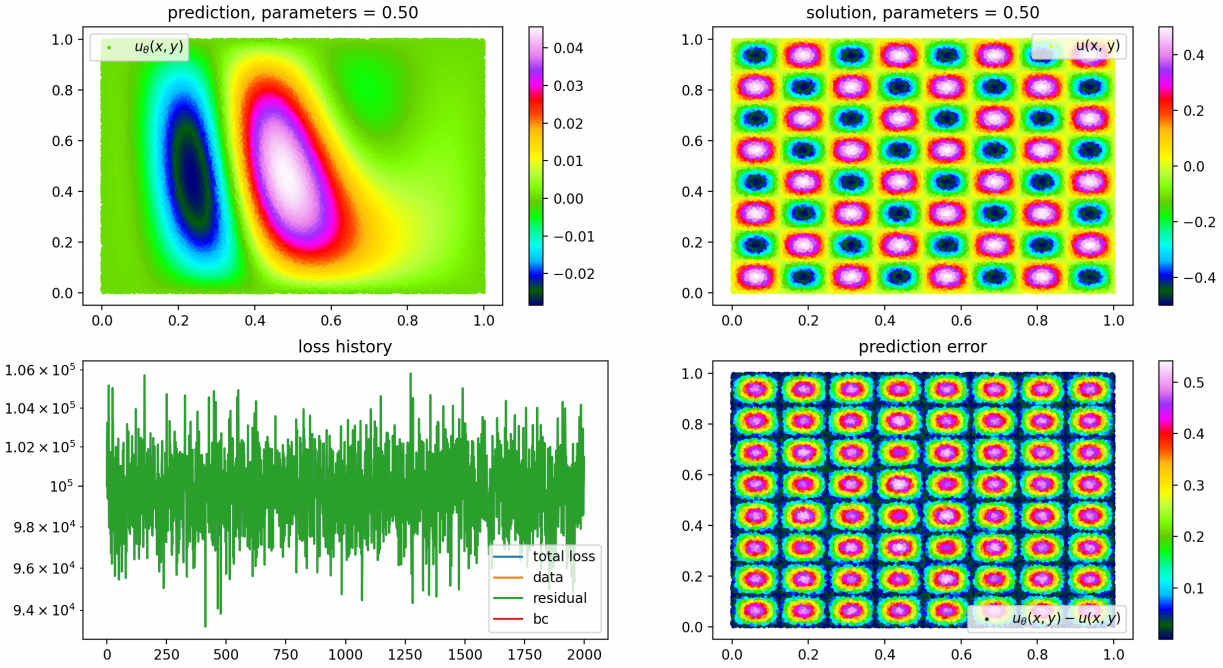
- Study $K_0(X, \bar{X})$ allows to understand the bias and fails of PINNs. For example some trouble arrive when a loss decay really faster than another.

Spectral biais ans high frequencies

Spectral bias

Using NTK we can study the **Spectral bias of MLP**. the MLP learn firstly the low frequencies and after the high frequencies (with difficulty)

- Classic MLP with Sinus activation function (to help). We solve $-\Delta u = 128 \sin(8\pi x)\sin(8\pi y)$



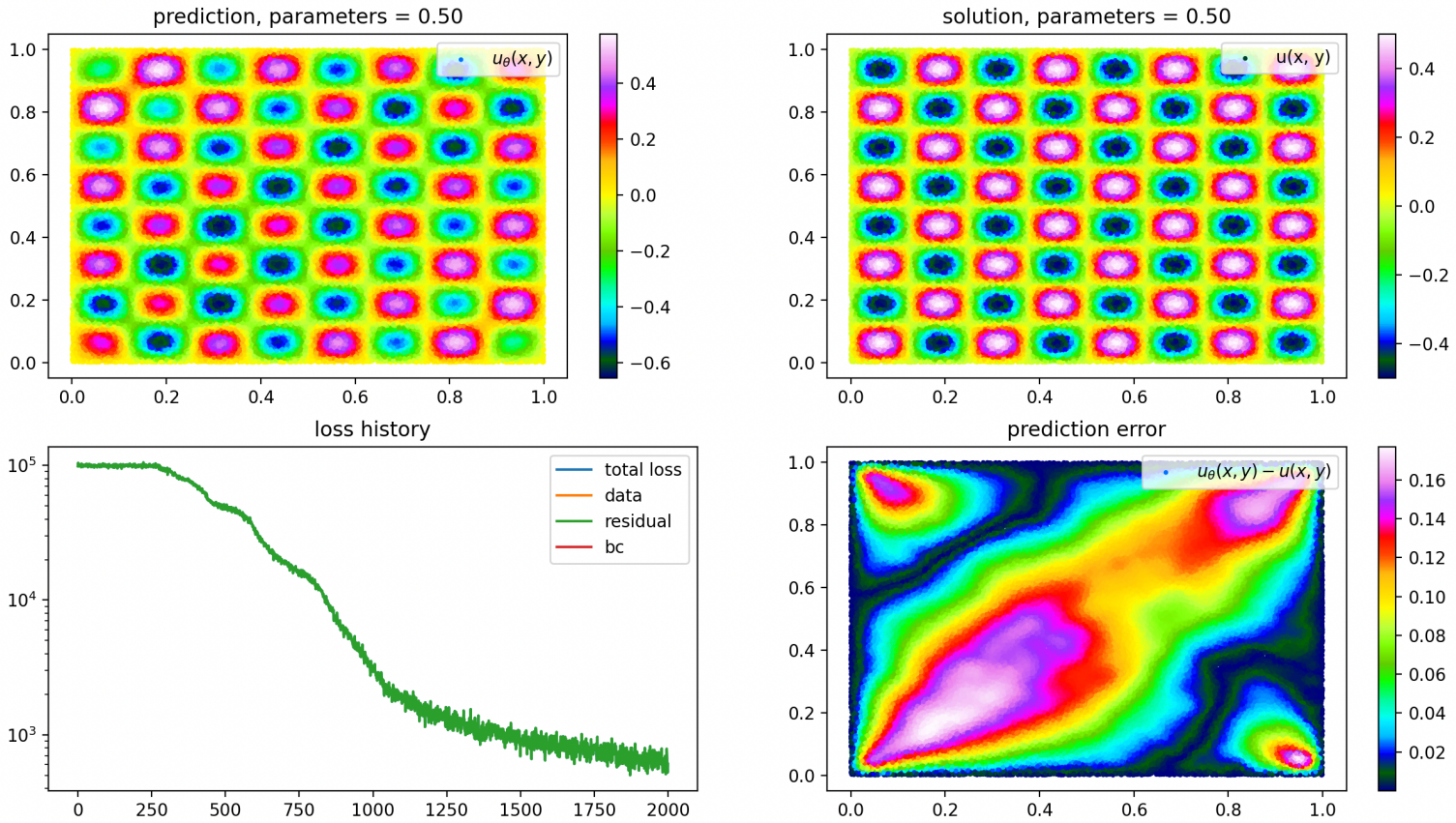
- To solve this problem for PINNs we add Fourier features. We replace

$$NN_\theta(x), \text{ by } NN_\theta(x, \sin(2\pi k_1 x), ..., \sin(2\pi k_n x))$$

with $(k_1, ...k_n)$ trainable parameters.

Spectral biais ans high frequencies II

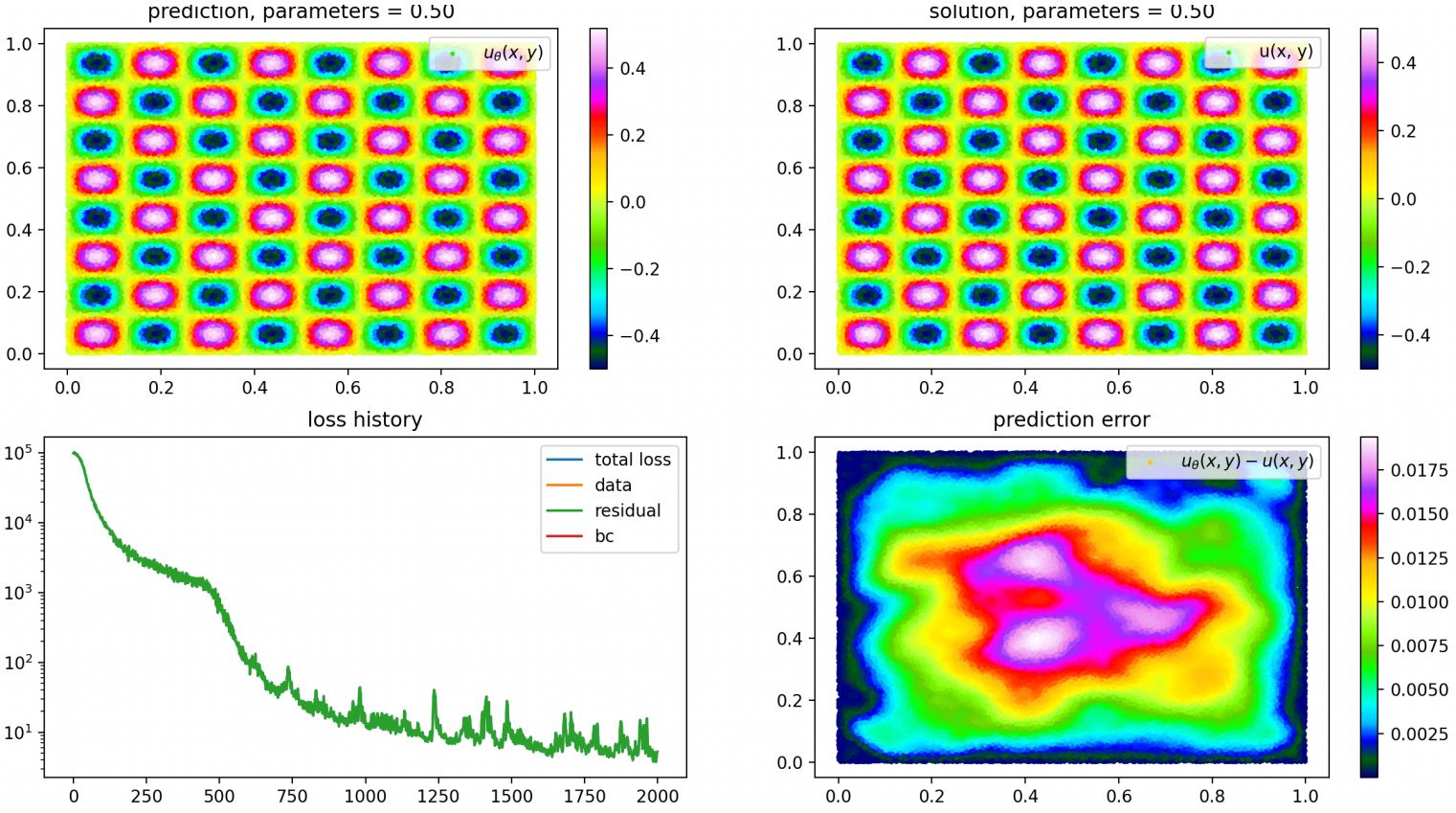
- Fourier Network with Tanh activation function.



- **Other interesting subjects**: adaptive sampling, loss balancing (to avoid bais) etc

Spectral biais ans high frequencies II

- MutiScale network with Tanh activation function.



- **Other interesting subjects**: adaptive sampling, loss balancing (to avoid bais) etc

Inverse problem and optimal control

Shape optimization I

- Since the PINNs use a minimization framework it will be easy to solve without large modification control optimal and inverse problems.
- Example: shape optimization.
- We introduce the energy associated to the Laplace problem for one domain:

$$E(\Omega) = \inf_{u \in H_0^1} \frac{1}{2} \left(\int_\Omega \mid \nabla u(x) \mid^2 - f(x)u(x)dx \right)$$

- A classical problem is to find the domain Ω which minimize this energy with a constrains volume :

$$\Omega^* = \inf_{\Omega, |\Omega| = V_0} E(\Omega)$$

Classical approach

Gradient method: we define a form gradient, we solve adjoint problem for that. Each change of domain needs a remeshing step. It is costly and non trivial.

Shape optimization II

Pinns approach

- We parametrize the PDE solution by a neural network $u_\theta(x)$,
- We consider a initial form Ω_0,
- We parametrize a mapping $m_\phi(x)$ such that $\Omega = m_\phi(\Omega_0)$.
- We solve:

$$\min_{\theta,\phi} \frac{1}{2}\left(\int_{m_\phi(\Omega_0)} \mid \nabla u_\theta(x) \mid^2 - f(x)u_\theta(x)dx\right)$$

- The integral is approximated with Monte Carlo approach. In practice we solve

$$\min_{\theta,\phi} \frac{1}{2}\left(\int_{\Omega_0} \mid \nabla(m_\phi(u_\theta(x))) \mid^2 - m_\phi(f(x))m_\phi(u_\theta(x))dx\right)$$

- There exist specific neural network called Sympnet which generate Symplectomorphism.

Idea

- In $\mathbb{R}^2$ the symplectomorphism preserve the volume. So we propose to use a SympNet for m_ϕ.

Shape optimization III

- PINNs on ellipse with hole.

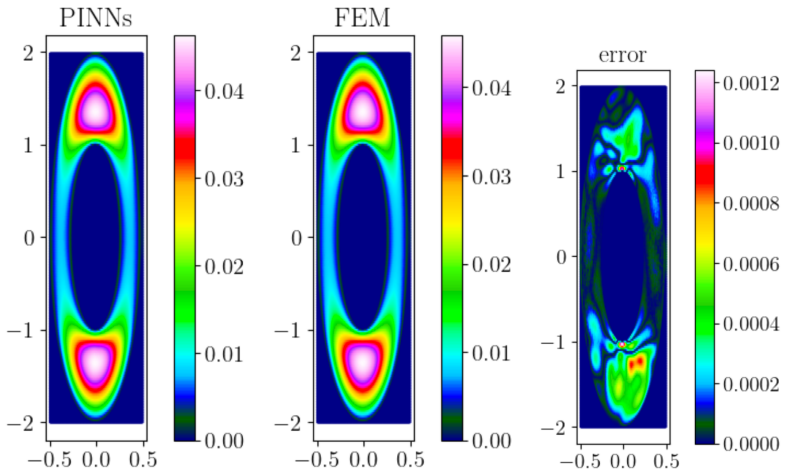


Fig. 4: Elliptic donut, $f = 1$

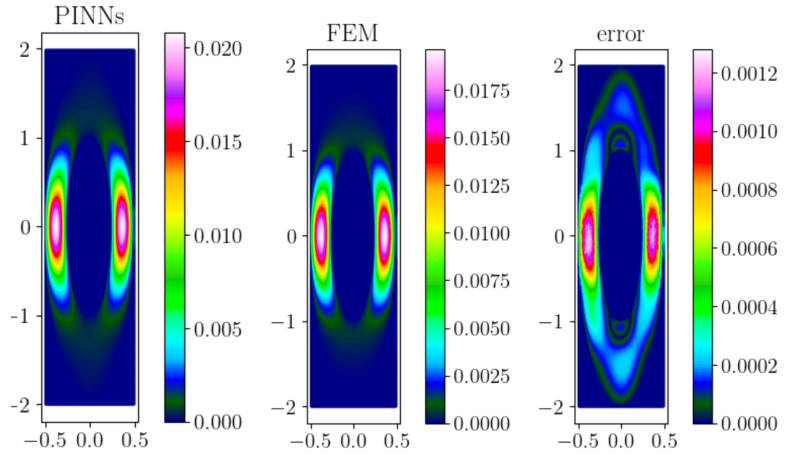
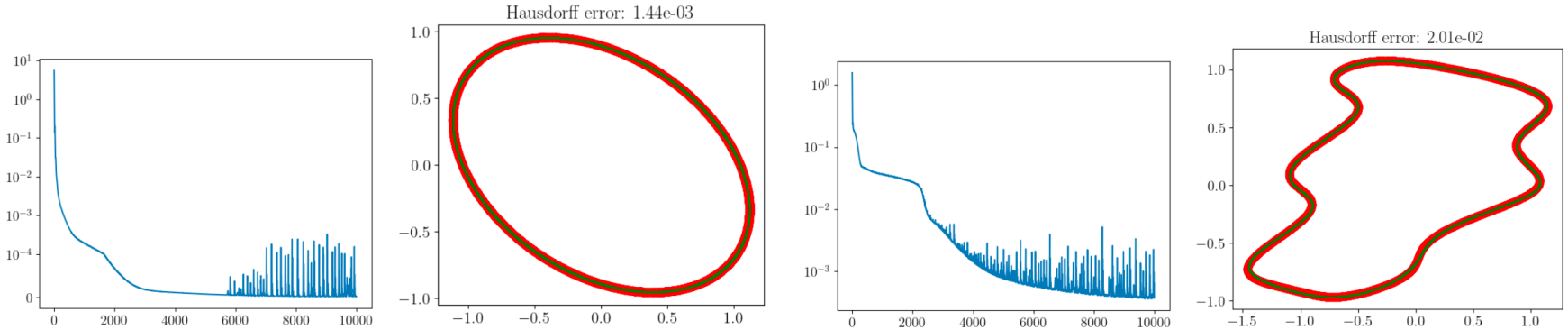


Fig. 5: Elliptic donut, $f = \exp(1 - r^2(x, y))$

- Learn mapping between shape:



Shape optimization IV

- Shape optimization with different sources

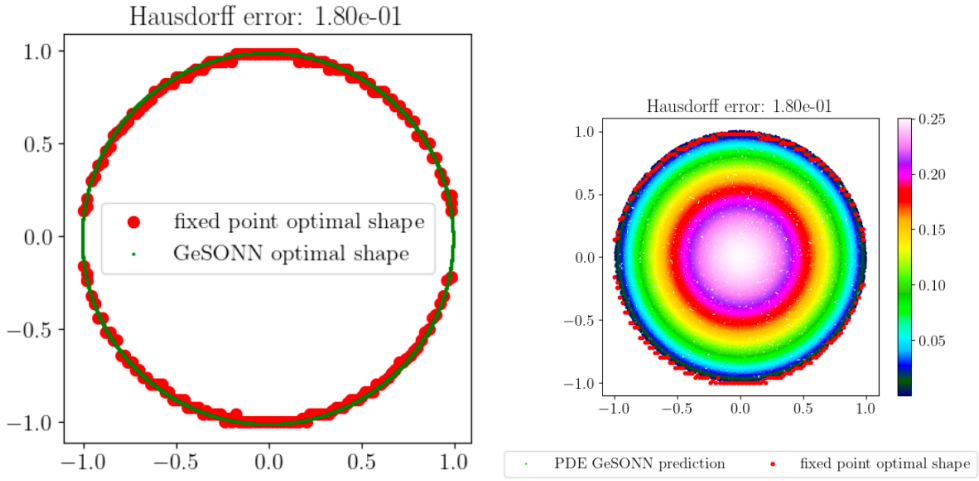
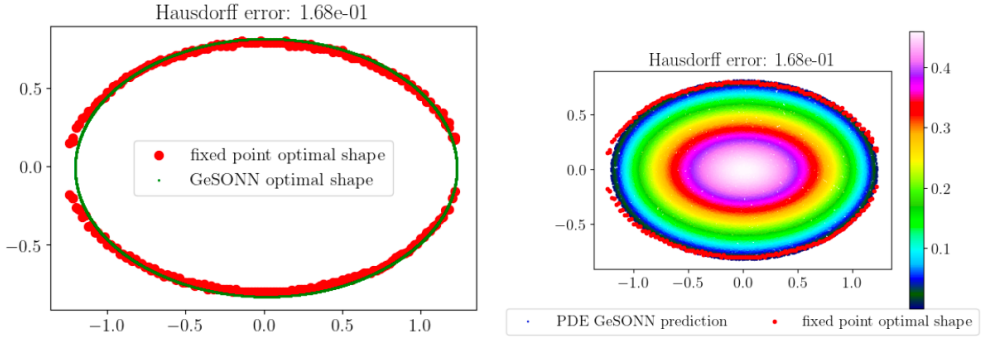


Fig. 10



- The initial shape is not a circle.

Conclusion

Conclusion

Short conclusion

- The PINNs and the Neural Galerkin can be view as a classical space time and space Galerkin approximation method where we project on a finite dimensional manifold (PINNs) or in the tangent space to the manifold (Neural Galerkin).
- We hope reduced significantly the number of parameters using manifolds. The neural networks seems good candidate for than in large dimensional input case.

Scimba

All the experiments of the talk have been realized with our library Scimba

Main references

- PINNs:
  - *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, M. Raissi, P. Perdikaris, G.E. Karniadakis
  - *An Expert's Guide to Training Physics-informed Neural Networks*, S. Wang, S. Sankaran, H. Wang, P. Perdikaris
  - *Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating PDEs*, S. Mishra, R. Molinaro
  - *Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks*, N. Sukumar and Ankit Srivastava
- Neural Galerkin:
  - *Neural Galerkin Scheme with Active Learning for High-Dimensional Evolution Equations*, J. Bruna, B. Peherstorfer, E. Vanden-Eijnden
  - *A Stable and Scalable Method for Solving Initial Value PDEs with Neural Networks*, M. Finzi, A. Potapczynski, M. Choptuik, A. Gordon Wilson
  - *Efficient Discrete Physics-informed Neural Networks for Solving Evolutionary Partial Differential Equations*,